Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Ančimer

# Uporaba okolju prijaznih tehnologij v velikih računalniških sistemih

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Jurij Mihelič
Somentor: doc. dr. Mojca Ciglarič

Ljubljana, 2016

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Uporaba okolju prijaznih tehnologij v velikih računalniških sistemih:

Okolju prijazno računalništvo oz. zeleno računalništvo je široko področje, ki pokriva vse od izdelave, uporabe in recikliranja računalnikov. V okviru diplomske naloge se osredotočite na tehnologije za varčno uporabo računalniških sistemov. Temeljito preglejte področje, predstavite načine proizvodnje električne energije iz obnovljivih virov in opišite mehanizme varčevanja z energijo v računalnikih in računalniških sistemih večjih razsežnosti, kot so superračunalniki, podatkovni centri in gruče računalnikov.

# Izjava o avtorstvu diplomskega dela

Podpisani Nejc Ančimer, z vpisno številko **63050002**, sem avtor diplomskega dela z naslovom:

*Uporaba okolju prijaznih tehnologij v velikih računalniških sistemih*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Jurija Miheliča in somentorstvom doc. dr. Mojce Ciglarič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 1. februarja 2016 Podpis avtorja:

Ljudem, ki jim je mar za okolje, v katerem živijo.

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **ACPI** | Advanced configuration and power interface | vmesnik za napredne nastavitve in napajanje |
| **ACTOR** | adaptive concurrency throttling optimization run-time system | optimizacijski sistem za dinamično omejevanje vzporednosti v času izvajanja |
| **CDM** | The Clean Development Mechanism | Mehanizem čistega razvoja |
| **CLFR** | compact linear Fresnel reflector | kompaktno linearno Fresnelovo zrcalo |
| **CSP** | concentrated solar power | koncentrirana sončna energija |
| **DCT** | dynamic concurrency throttling | dinamično omejevanje vzporednosti |
| **DVFS** | dynamic voltage and frequency scaling | dinamično skaliranje napetosti in frekvence |
| **ERP** | energy-reliability product | produkt energije in zanesljivosti |
| **IPC** | instructions per cycle (in section 3.4), interleaved parity check (in section 3.6.1 ) | ukazi na (urin) cikel (v razdelku 3.4), prepleteni paritetno preverjanje (v razdelku 3.6.1) |
| **IRAW** | idle read after write | preverjanje po vpisu med neaktivnostjo |
| **LSE** | latent sector error | latentna okvara sektorja |
| **MAID** | massive array of idle disks | masivno polje neaktivnih diskov |
| **MIPS** | millions of instructions per second | miljon ukazov v sekundi |
| **MLET** | mean latent error time | povprečni čas latentne okvare |
| **MRT** | mean response time | povprečni čas za odgovor |
| **MTTDL** | mean time till data loss | povprečni čas do izgube podatkov |

| | | |
|---|---|---|
| **PDC** | popular data congestion | zgoščanje popularnih podatkov |
| **RAID** | redundant array of inexpensive disks | polje poceni diskov z (podatkovno) odvečnostjo |
| **SSD** | solid-state disk | trdo-ožičeni disk |

# Povzetek

Cilj diplomske naloge je pregledati trenutne zelene informacijske tehnologije in trenutne raziskave na omenjenem področju. Osredotoča se na teme varčne uporabe računalniških sistemov večjih razsežnosti. Za začetek poskuša podati razlog za uvedbo varčevanja z energijo s temami iz področja podnebnih sprememb. Nato se osredotoča na energetsko učinkovitost računalniških sistemov večjih razsežnosti. Neuspeh je v dejstvu, da diploma ne bo rešila problema podnebnih sprememb. Za boljšo energetsko učinkovitost že poznamo mehanizme za varčevanje z električno energijo s prenosnih naprav. Ti mehanizmi potrebujejo prilagoditve za uporabo na večjih računalniških sistemih kot so superračunalniki in podatkovni centri. Uspeli smo najti in predstaviti nekaj primerov uporabe mehanizmov varčevanja za računalniške sisteme večjih razsežnosti.

**Ključne besede:** emisije, podnebne spremembe, globalno segrevanje, zeleno računalništvo, okolje, energija, računalniški sistem, poraba energije, prilagajanje, uravnavanje frekvence, uravnavanje napetosti, energetska učinkovitost, varčevanje z energijo, procesor, Linux.

# Abstract

The aim of the thesis is to review current green information technologies and current research on the mentioned area. It's focused on themes of efficient usage of large scale computer systems. At the beginning it tries to give a reason for implementation of energy savings with themes from area of climate change. Afterwards it's focused on energy efficiency of large scale computer systems. The failure is in fact that this thesis is unable to solve the problem of climate change. For a better energy efficiency we already know the mechanisms of electrical energy savings from mobile devices. These mechanisms need an adaptation to be useful for a large scale computer systems as in supercomputers and data centres. We were successful in finding and presenting of a few usage examples for saving mechanisms on a large scale computer systems.

**Keywords:** emissions, climate change, global warming, green computing, environment, energy, computer system, energy consumption, adapting, frequency scaling, voltage scaling, energy efficiency, energy savings, processor, Linux.

# Poglavje 1

# Uvod

## 1.1 Motivacija

V času študija sem naletel na dokumentarni film Neprijetna resnica (*An Inconvenient Truth*). V tistem času se je problem imenoval "globalno segrevanje". Predstavili so predvsem posledice, kot so taljenje ledenikov po svetu, bolj ekstremno vreme in selitve živali in ljudi zaradi višjih temperatur in dviga morske gladine. Torej predvsem z vidika podnebja. V filmu razložijo, da se svet segreva zaradi učinka tople grede, za katerega človek proizvaja velike količine izpustov toplogrednih plinov z uporabo fosilnih goriv.

Na presenečenje znanstvenikov in okoljevarstvenikov je bil problem prezrt. *Intergovernmental Panel on Climate Change (IPCC)* obstaja že od leta 1988. Na problem podnebnih sprememb so opozarjali že od leta 1990 v njihovem prvem poročilu [49]. Do danes je bil problem odlagan in vsi ukrepi za zmanjšanje uporabe fosilnih goriv so bili premalo učinkoviti, med njimi tudi neuspeli mednarodni sporazum v mestu Copenhagen leta 2009 [20]. Z leti spoznavamo podnebne spremembe in posledice uporabe fosilnih goriv tudi iz drugih področij.

Kitajci vse bolj spoznavajo fosilna goriva preko vdihavanja finih delcev, manjših od 2,5 mikrometra - PM2,5 (in manjših), ki nastanejo kot posledica izgorevanja (večinoma neprečiščenega) oglja in (nizko-kakovostnega) goriva za vozila. Poznamo jih tudi pod imenom črni ogljik (*black carbon*). So v smogu, ki se pogosto dviga nad mesti in prometnejšim podeželjem. Povzroča pljučna obolenja in raka na

pljučih. Vse to izvemo iz dokumentarnega filma Pod kupolo (*Under The Dome*), katerega ogled je kitajska vlada skušala preprečiti. Ogled na spletni strani Youtube je mogoč, vendar je dostop iz Kitajske blokiran [8]. Posledice so na zdravju ljudi in tudi njihova pričakovana življenjska doba se je znižala za pet in pol let [13].

Papež Frančišek in vatikanskimi znanstveniki so opozoril tudi na socialni pogled na podnebne spremembe. Predvsem njihov vpliv na pridelavo hrane [17]. In dolg razvitih držav do revnih dežel sveta. Želeli so poudariti, da se prvi s podnebnimi spremembami že spopadajo revni in dodati moralni poziv za akcijo proti podnebnimi spremembami in razvoj revnejših dežel sveta [14].

Kljub vse več negativnih vplivov in s tem v resnici večji ceni uporabe fosilnih goriv ni opazen množičen premik proč od fosilnih goriv. Problem je bil spolitiziran, politiki in vplivni ljudje so zanikali njegov obstoj. Nekateri so ga priznavali, vendar so zanikali človekov vpliv nanj. Gre za ščitenje industrije fosilnih goriv kot se je že zgodilo v preteklosti s cigaretami. Več o tem lahko izveste iz knjige Prodajalci dvoma (*Merchants of Doubt*) ali enako naslovljenega dokumentarnega filma.

Ogled dokumentarnega filma iz leta 2009 Naš dom (*Home*) pojasni, da so mnogi problemi, med njimi tudi podnebne spremembe povezane med seboj in z našim načinom življenja. Še bolj zanimiv je film Opravite računanje (*Do the Math*), ki nam pove da, če želimo svet leta 2100 segret za povprečno globalno letno temperaturo 2°C, smemo spustiti v zrak 565 milijard ton $CO_2$ (iz leta 2012, podatek je že strožji). Rezerva industrije fosilnih goriv v tistem času je ocenjena na 2795 milijard ton $CO_2$. Če torej izčrpamo in uporabimo vso to zalogo nas čaka leta 2100 svet, ki bo od 4°C do 6°C toplejši (povprečno globalno, lokalne temperature so težje določljive). Mnogi se bojijo, da bodo klimatske spremembe postale nevarne, med drugim tudi sposobne napajati same sebe. To je tudi najhujši možni scenarij.

Priznati moram osebni interes za to področje. Začel sem brez znanja o varčevanju z energijo na računalniških sistemih. V tej diplomski nalogi bom poskusil identificirati mehanizme, ki računalniškim sistemom omogočajo varčevanje s potrošnjo električne energije. Omenili bomo tudi uporabo obnovljivih virov, ki zmanjšajo naše breme na okolje.

## 1.2 Ogljični odtis

Ogljični odtis je meritev vsote izpustov ogljikovega dioksida in metana, ki ga opazovana organizacija, sistem ali aktivnost povzroča ob upoštevanju proizvodnje, odstranitve in shrambe (omenjenih dveh) plinov, vendar prostorsko in časovno omejena na opazovano organizacijo, sistem ali aktivnost [27]. Enota za to mero je ekvivalentna masa ogljikovega dioksida, ki lahko zajame enako količino toplote v obdobju 100 let, kot bi jo mešanica različnih toplogrednih plinov. Merjenje in preračunavanje v ekvivalent ogljikovega dioksida se uporablja predvsem zaradi prepoznavnosti tega toplogrednega plina.

Ker ob proizvodnji iz fosilnih goriv nastaja energija in izpusti na oddaljenih lokacijah termoelektrarn, ogljični odtis merijo na zgradbah kot količino potrebne električne energije. Vozila izpuščajo toplogredne pline iz pogona na notranje izgorevanje, zato ogljični odtis predstavlja količina porabljenega goriva. Vir metana so tudi organski odpadki vseh vrst, torej tudi količina proizvedenih odpadkov povečuje ogljični odtis.

Pri Kjotskem protokolu prepoznavajo in želijo omejiti tudi druge toplogredne pline. Pri tem ohranijo enoto, uporabljeno pri ogljičnem odtisu. Uporablja se odtis toplogrednih plinov (*greenhouse gas footprint - GHG footprint*). Količine za celotno državo so v milijonih ton ekvivalenta ogljikovemu dioksidu. Za vsak posamezni plin se masa izrazi v teragramih, nato zmnoži s toplogrednim potencialom za stoletno obdobje oziroma z $GWP100$, ki je konstanta. Formula za odtis toplogrednih plinov (in ogljični odtis):

$$m_{CO_2eq} = \sum_{i=1}^{n} (m_i * GWP100i) \tag{1.2.1}$$

V enačbi 1.2.1 $n$ predstavlja število toplogrednih plinov, kolikor jih imamo v mešanici. Masa posameznega plina ($m_{(i)}$) je pretvorjena preko plinovega pripadajočega toplogrednega potenciala za stoletno obdobje, $GWP100$, v ekvivalentno maso ogljikovega dioksida, $m_{CO_2eq}$, z enakim učinkom v stoletnem obdobju.

Na splošno so področja glede na količino izpustov za posameznega človeka razvrščena v sledečo lestvico [27] (z nekaj primeri):

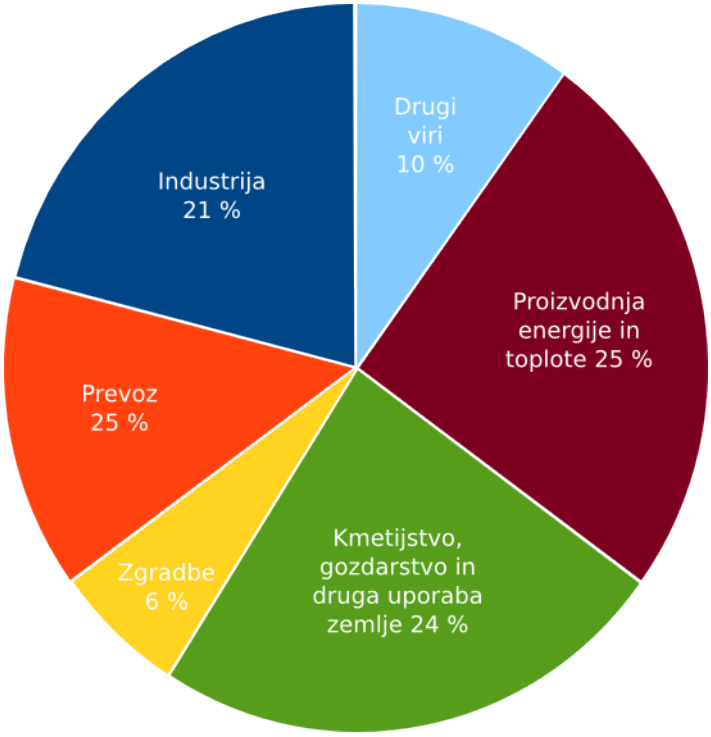
1. Transport: predvsem avtomobili, dnevna migracija na delo, poslovni leti z letali in manjši prispevki iz javnega prevoza naredijo to področje največji vir izpustov toplogrednih plinov.

2. Nepremičnine: uporaba električne energije za gradnjo, elektronske naprave, hlajenje, ogrevanje, razsvetljava.

3. Hrana: Največ izpustov prispeva živinoreja. Predvsem goveda za meso in mlečne izdelke. Sem spada tudi pridelava drugih vrst hrane z nižjimi izpusti v primerjavi z govedorejo.

4. Izdelki.

5. Storitve. Brskanje po spletu torej prispeva najmanj iz te lestvice.




Slika 1.1: Globalni izpusti leta 2010 iz poročila *IPCC* leta 2014. Narisano po viru [65].

Računalnik se v gospodinjstvu praktično izgubi med ostale električne naprave in razsvetljavo prostorov in ima izredno majhen samostojen ogljični odtis. Če pogledamo globalnega na sliki 1.1, je sektor informacijskih tehnologij absorbiran v proizvodnji energije in toplote. Glede na grobo oceno naj bi celoten sektor informacijskih tehnologij predstavljali 2 % do 2,5 % svetovnih izpustov [10]. Zajema

domače računalnike, zaslone, mobilne in fiksne komunikacijske platforme in podatkovna središča. Vendar pričakujejo, da bo ta sektor narasel zaradi priljubljenosti mobilnih naprav in storitev podatkovnih središč in superračunalnikov.

## 1.3 Zmanjševanje ogljičnega odtisa

Kot posamezniki lahko manj potujemo. Uporabimo električni avto, kolo ali javni prevoz. Izboljšamo izolacijo nepremičnin, manj ogrevamo, se bolje oblečemo in manj hladimo. Pri hrani jemo manj mesa in poskušamo pridobiti lokalno pridelano hrano. Pozanimamo se in kupujemo električne naprave, ki zahtevajo manj energije za delovanje. Ostale rešitve zahtevajo zeleno, ekološko oziroma, kakor se imenuje sedaj, trajnostno gradnjo. S pravilno gradnjo za uporabo nepremičnine potrebujemo zelo malo dobrin in storitev izven nepremičnine.

Za računalniška podjetja in zaposlene velja enako kot v primeru posameznika. Ogljični odtis bi lahko znižali na področju nepremičnin s pravilno izbiro lokacije, kjer je na primer vedno hladno in niti ne bi potrebovali aktivnega hlajenja. Drugo je področje energetsko učinkovite izdelave, kjer kot kupec izberemo ustrezno energetsko učinkovito, vendar še vedno ustrezno zmogljivo strojno opremo. Zadnje je področje storitev, za katero bi uporabljali optimizirano programsko opremo, vendar podjetja običajno uporabljajo tuje programske rešitve, ki praviloma niso optimizirane točno za namene podjetja. Rešitev je torej tudi v samostojnem razvoju programskih rešitev, ki delajo minimalno točno tisto, kar podjetje namerava ponujati.

## 1.4 Kjotski protokol

Kjotski protokol je mednarodni sporazum, s katerim želijo države formalno, s predpisi omejiti izpuste toplogrednih plinov. Izpuste toplogrednih plinov morajo omejiti le industrijske države, članice protokola, katerim postavijo dovoljeno mero izpustov, tako imenovane “ogljične kredite” oziroma dovoljeno maso izpustov. Pri tem ne upoštevajo izpustov iz letalskega prometa ali javnega prometa, ki so torej opravičeni. Toplogredni plini, za katere omejujejo izpuste so, v tabeli 1.1 [50]:

Tabelo smo oblikovali, da olajšamo iskanje po virih v angleškem jeziku. Dodali

| vrsta plina | angleško | formula | GWP100 |
|---|---|---|---|
| ogljikov dioksid | *carbon dioxide* | $CO_2$ | 1 |
| metan | *methane* | $CH_4$ | 21 |
| dušikov oksid | *nitrous oxide* | $N_2O$ | 310 |
| žveplov heksaflourid | *sulphur hexafluoride* | $SF_6$ | 23900 |
| fluorirani ogljikovodiki | *hydrofluorocarbons* | $C_?H_?F_?Cl_?$ | 140 - 11700 |
| fluoroogljiki | *perfluorocarbons* | $C_?F_?Cl_?$ | 6500 - 9200 |

Tabela 1.1: Tabela toplogrednih plinov

smo tudi kemijsko formulo, ker so včasih plini omenjeni le v takšni obliki. In dodali smo jim tudi pripadajoči toplogredni potencial za stoletno obdobje. Zadnji dve vrstici v tabeli predstavljata skupino, množico plinov, zato ni konkretne formule. Navedeni so le elementi, ki bi jih v posameznem predstavniku te skupine plinov lahko našli, ker na primer klora ne najdemo v vseh toplogrednih flouroogljikih. In ker gre za skupino je tudi pripadajoči toplogredni potencial območje vrednosti od minimalne do maksimalne.

Države lahko izpuste znižajo preko izvedbe projektov v okviru "Mehanizma čistega razvoja" (*Clean Development Mechanism* - CDM). Države z zastavljenimi cilji oziroma omejitvami izpustov imajo na voljo še "Združeno implementacijo" (*Joint implementation* - JI). V obeh primerih gre za formalnost, kjer na podlagi predloge projekta v zameno posamezniki ali organizacije prosijo za priznanje znižanja izpustov in ob potrditvi dobijo "ogljične kredite". Če jim izpusti še vedno presegajo dodeljene "kredite", jih lahko kupijo od drugih držav, ki so cilje že uspele doseči (ali jih nimajo) in imajo odvečne ogljične kredite.

Večina je skeptičnih glede učinkovitosti Kjotskega protokola, saj bi za ohranitev količine toplogrednih plinov v ozračju morali postaviti strožje omejitve izpustov. Omogoča preveč prilagodljive mehanizme. Na primer države lahko dosežejo cilje preko trgovanja s krediti, ki jih je preveč in imajo zato glede na ponudbo in povpraševanje nizko ceno. Državam je enostavneje trgovati namesto dejansko znižati količino izpustov.

Problemi so nastali tudi s samimi projekti, katerih dejansko zmanjšanje izpustov je vprašljivo. Uradniki, ki odločajo o sprejemljivosti projekta po Kjotskem

protokolu, priznavajo zmanjšanje in ne kaznujejo ob izvajanju projekta dodatnih ali kasneje nastalih izpustov. Protokol je preveč popustljiv. Cilje bi morale imeti tudi države v razvoju. Kitajska ni obravnavana kot industrijska, torej nima omejitev, vendar zadnja leta vodi na lestvicah državnih izpustov $CO_2$. Cilje bi prevzela po letu 2020. Na mednarodni konferenci za podnebne spremembe v Parizu 2015, so države sporočale svoje prostovoljne prispevke k znižanju izpustov brez uporabe kjotskega protokola.

Novinarska agencija Reuters [11] in poročilo "Podatkovne baze emisij za globalno raziskovanje atmosfere" (*Emissions Database for Global Atmospheric Research* - EDGAR [19]) navajajo vsakoletno povečano količino izpustov toplogrednih plinov. Zaključimo lahko torej, da se "apetiti" (ne potrebe) po energiji povečujejo, zato ni videti stabilizacije vsebnosti ogljika v ozračju.

Kaj to pomeni za podatkovni center? Preko CDM bi na lokalni upravi zaprosili za priznanje znižanja izpustov zaradi pogozditve izbranega območja. Pridobili bi ogljične kredite. Nato bi uradniki ugotovili, da nameravamo pogozditi že pogozdeno območje. Podobno se je že dogajalo v preteklosti, zato so lokalni uradniki spremenili pogoje za sprejetje projektov. Drugi primer je, da na primer na Kitajskem, ki nima ciljev, nekdo drug izvede projekt. Izvedba podeli kredite izvajalcu na Kitajskem, ki jih nato postavi na trg, namenjen ogljičnim kreditom. Mi te kredite kupimo. V prvem primeru smo imeli delo le s papirji, v drugem smo zapravili malo denarja. V obeh primerih se na samem podatkovnem centru ni spremenilo nič.

## 1.5 Pregled vsebine

Naslednje poglavje 2 je namenjeno proizvodnji električne energije iz obnovljivih virov energije. Trenutno najbolj prepoznavna predstavnika teh virov sta sončna in vetrna energija, vendar smo grobo predstavili tudi ostale. Pod vsako obliko smo opisali, kako jo izkoriščamo, in našteli nekaj splošnih problemov z njimi.

Sledi poglavje za računalniške teme iz večje energetske učinkovitosti. V prvem razdelku 3.1 začnemo na strojnem nivoju in opišemo tehnike za omejevanje porabe električne energije v vezju. Nato predstavimo tehnologije za prihodnjo generacijo procesorjev, ki bodo znižale tudi porabo.

Izmed naštetih tehnik za varčevanje z obstoječo tehnologijo se bomo nato osredotočili predvsem na upravljanje frekvence procesorja. Zaradi prevladovanja tem okoli te tehnike smo namenili praktičen razdelek 3.2, kako se z nastavitvami v Linuxu lahko uporabi ta mehanizem za varčevanje. Izvedli smo tudi praktičen poskus, da bi dokazali smiselnost uporabe tega mehanizma za varčevanje s porabo električne energije.

Sledi razdelek 3.3, namenjen algoritmu za avtomatično izbiro frekvence procesorja. Algoritem za prilagajanje frekvence na podlagi spremenljivke $\beta$ se osredotoča predvsem na merjenje aktivnosti na vhodu/izhodu (strojni dogodki) preko količine opravljenih ukazov vzporedne aplikacije, da razpozna priložnosti za znižanje frekvence procesorja.

Naslednji razdelek 3.4 za prepoznavanje ustrezne frekvence procesorja izrablja strojno učenje. Nadzoruje tudi število in razpored niti k procesorjem, vendar je omejen na vzporedne aplikacije, ki uporabljajo knjižico OpenMP.

V naslednjem razdelku 3.5 se posvetimo strežnikom. Ne osredotočamo se na posamezne rešitve za uravnavanje porabe, temveč poskušamo predstaviti sistem, ki bi vzpostavil komunikacijo med temi rešitvami in jih uskladil, da bi se izognili konfliktom med odločitvami posamezne rešitve. Opisana je kompleksnost problemov in le predlog za njihovo rešitev.

Tokrat razdelek 3.6 namenimo hranjenju podatkov. Na trdih diskih vse bolj potrebujemo zanesljivost, vendar tudi varčevanje s porabo električne energije. Zaradi pomanjkanja študij združevanja zanesljivosti z varčevanjem le naštejemo tehnike s samostojnih področij. Nato predstavimo začetek z vzpostavitvijo merila za uspešnost združevanja tehnik zanesljivosti in varčevanja.

Zadnji razdelek 3.7 je namenjen predlogom za uporabo prevajalnikov za varčnejšo izrabo shrambe podatkov, ki pri dinamičnem prevajalniku ni omejena le na trde diske. Zadnji predlog se osredotoči na varčevanje v komunikaciji za arhitekture omrežij na čipu.

# Poglavje 2

# Obnovljivi viri energije

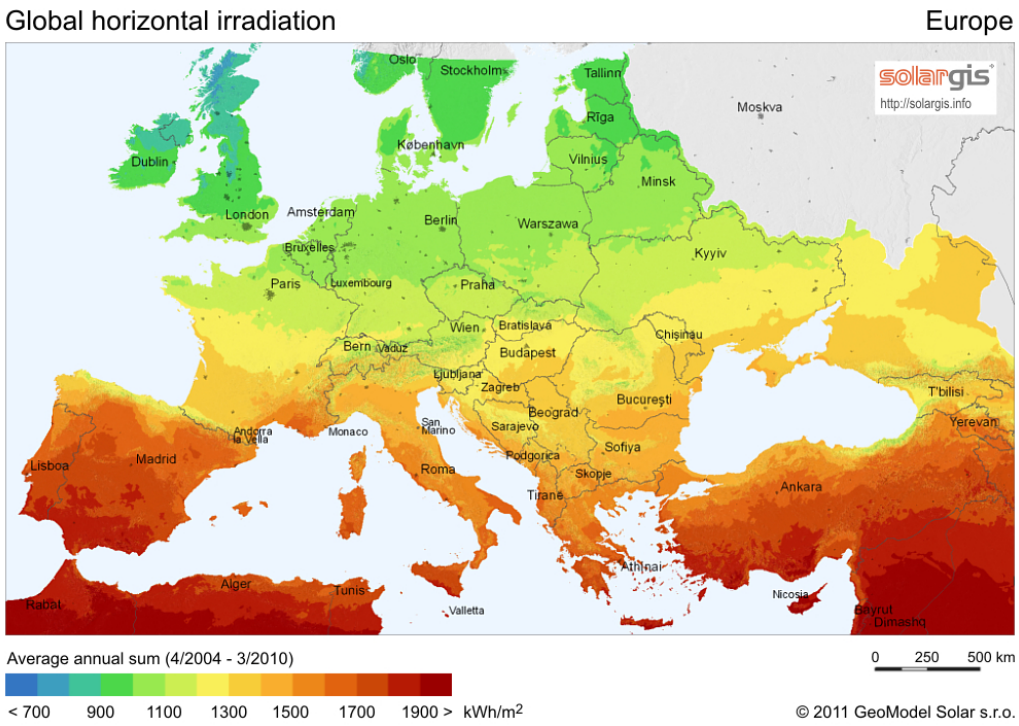
Problema klimatskih sprememb niti ne bi bilo, če bi uporabljali zelene oziroma obnovljive vire energije. Sežig biomase izpušča toplogredne in zdravju škodljive strupene pline, vendar ogljik, ki je v njih, je bil s fotosintezo vezan na njihova telesa iz zraka. Povedano drugače, bil je že v zraku. Fosilna goriva za razliko od biomase črpamo iz zemlje. Ogljik, ki je bil zakopan pod zemljo, torej dodajamo k že obstoječemu v zraku.

Vsi obnovljivi viri sicer niso vedno na voljo, kar je glavni očitek tem vrstam energije. Vendar tudi ta očitek bledi z inovacijami na področju hranjenja energije [18].

## 2.1 Sončna energija

Sončna energija nosi veliko količino energije, ki jo koristijo večinoma rastline. V naših krajih je ocenjena na okoli 1200 kWh/m$^2$ v celotnem letu. Veliko oblik sončnih elektrarn je začelo kot eksperiment, ki je potem uporabljen komercialno.

Energijo sonca koristimo na dva načina. Prvi način s pomočjo zrcal preusmerja sončno svetlobo z večje površine v skupno točko, v kateri se svetloba zbira. Tam svetlobo prestrežejo in jo uporabijo za segrevanje vode, ki s segrevanjem izpari. Para nato poganja parno turbino, kjer se toplota pretvori v elektriko. Ta način je znan pod imenom koncentrirana sončna energija (*concentrated solar power*) ali koncentrirana sončna toplota (*concentrated solar thermal*). Problem pri tem pristopu je predvsem v neučinkovitosti pretvorbe iz toplotne v električno energijo.




Slika 2.1: Ocena letne proizvodnje sončne energije za Evropo [61]. Avtor: SolarGIS © 2011 GeoModel Solar s.r.o. Po licenci: [36]

Drugi način neposredno pretvori sončno svetlobo v električno energijo s pomočjo foto-električnega efekta. To dosežejo s fotovoltaičnimi celicami, ki jih vežejo skupaj v module. Na teh dveh pristopih so se razvile sedanje sončne elektrarne z nekaj razlikami med njimi [31, 63].

Vse vrste sončnih elektrarn imajo skupne nadloge. Prah, pesek, posušene odtise dežja, sneg, zamegljenost ali druge vrste nečistoč, ki ovirajo zbiranje ali zrcaljenje sončne svetlobe. Prepuščene so vremenskim razmeram.

Idealno bi sončno svetlobo zajeli v vesolju in pošiljali na več sprejemih postaj na zemlji, da omogočimo celodnevni sprejem energije. Vendar so stroški, povezani z izdelavo satelitov in zemeljskih sprejemnih postaj za celodnevni sprejem preveliki. Problem so tudi ostali odpadki v orbiti in nezmožnost vzdrževanja satelitov. Našteto ohranja to obliko sončne elektrarne kot idejo. Teoretično bi lahko 90600 $km^2$ veliko območje Sahare namenili sončnim elektrarnam in z njimi nadomestili vse elektrarne sveta [54].

Slika 2.2: Parabolična zrcala [41]. Avtor (domnevni): Kjkolb. Po licencah: [29, 36]

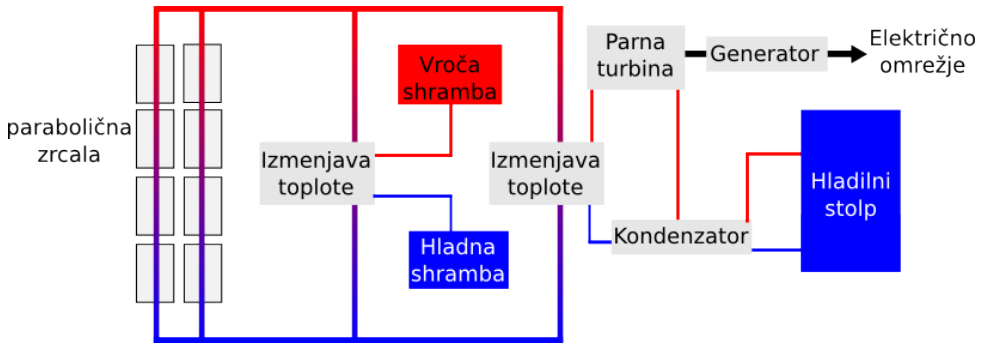
### 2.1.1 Parabolična zrcala

Standardna oblika sončne elektrarne zbira svetlobo na podolgovati cevi, pod katero so postavljena parabolično zakrivljena zrcala. Ta zrcala in cev za zbiranje svetlobe so premična, saj se s premikom zrcala spremeni položaj točke zbiranja svetlobe nad paraboličnim zrcalom. Potrebuje sledenje soncu le v eni osi. Svetloba se zbira vzdolž cevi, vedno v zbirni točki nad parabolo, zato sledenje v dveh oseh ni potrebno. Toplotna energija, zbrana v cevi, se prenese do parne turbine, kjer se pretvori v električno [31, 51].

Glede na starost te oblike in množičnost uporabe je to najbolj preizkušena oblika. Dosega okoli 20 % učinkovitosti pretvorbe iz sončne svetlobe v električno energijo [32]. Izboljšave tega modela so v zrcalih, tekočini za prenos toplote in načinu zbiranja svetlobe.

#### Shranjevanje toplote

S shranjevanjem toplote v kotlih omogočijo, da elektrarna lahko ustvarja elektriko ponoči in v primeru slabega vremena. Na tem področju lahko pričakujemo evolucijske napredke časa hranjenja z uporabo boljših izolacijskih materialov ali




Slika 2.3: Model sončne elektrarne s paraboličnimi zrcali. Narisano po viru [52].

uporabo snovi, ki imajo boljšo termalno maso. Prvi modeli sončnih elektrarn niso omogočali hranjenja proizvedene električne energije ali toplote in so bili torej ponoči ali ob slabem vremenu neuporabni.

### Tekoča sol

Namesto vode se za prenos toplote uporablja tekočo sol. Tekoča sol zadrži večjo količino toplote, torej ima boljšo termalno maso. Primerne so soli iz fluorida, klorida ali nitratov. Prihodnost vidijo v hidriranih soleh, torej mešanicah soli in vode v nekem razmerju. Uporablja se parna turbina, torej pred njo izmenjajo toploto med tekočo soljo in vodo, ki izpareva. Po izhodu iz turbine morajo paro ohladiti in kondenzirati nazaj v vodo. Za to se običajno uporabi hladna voda iz okolice [48, 31, 63, 58].

### Kompaktni linearni Fresnelov reflektor

Kompaktni linearni Fresnelov reflektor (*CLFR*) uporablja manjša vzporedna ravna ali rahlo zakrivljena zrcala. Vsakega lahko obračajo ter s tem dosežejo večjo odbojno površino v primerjavi z velikim paraboličnim, ki bi zavzel enak prostor. Ravna zrcala so tudi cenejša in ne potrebujejo premičnega sprejemnika svetlobe, zato je *CLFR* lažje izdelati. Sprejemnik, v katerem je več manjših cevk, je na fiksni točki nad skupino zakrivljenih zrcal. Nekateri modeli uporabljajo dva sprejemnika, kjer zrcala izmenično preusmerjajo svetlobo v prvega in drugega. Kljub prednostim niso v širši uporabi [30].

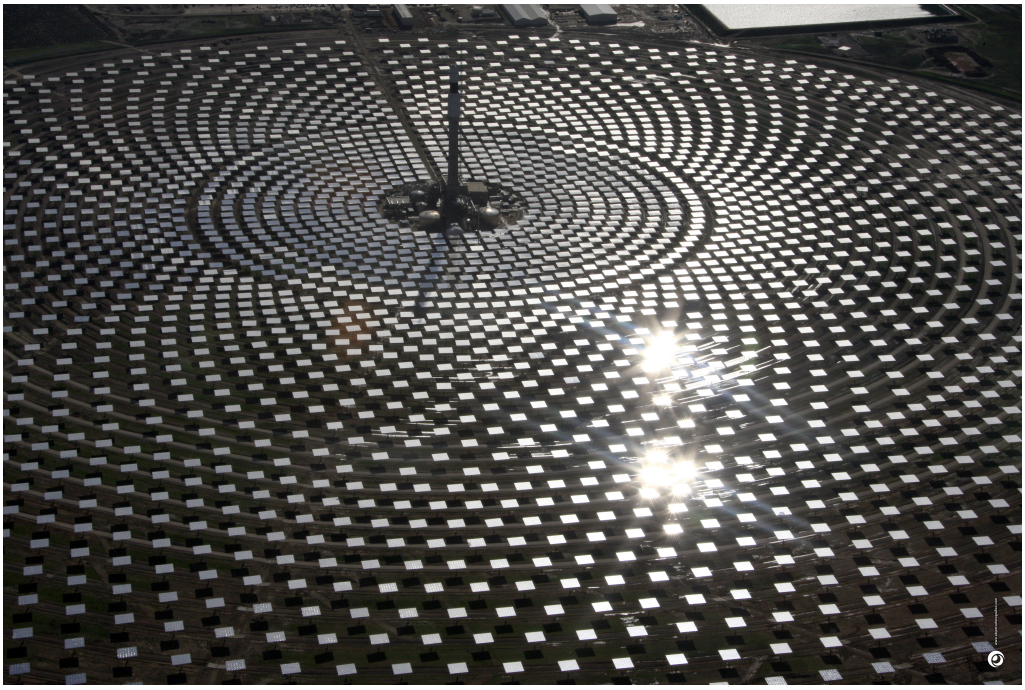

Slika 2.4: Elektrarna iz kompaktnih linearnih fresnelovih reflektorjev [40]. Avtor: Novatec Solar. Po licenci: [36].

### 2.1.2 Solarni stolp

Solarni stolp je oblika sončne elektrarne, pri kateri se zbira vsa svetloba v sprejemniku na vrhu solarnega stolpa, od koder zbrano toploto shranijo v shrambi za segreto tekočo sol. Odbojna zrcala so razporejena na ravnini in del odbite svetlobe lahko prestrežejo zrcala na poti do vrha stolpa. Stroške lahko zmanjšajo z izbiro ravnih zrcal. V puščavah vode primanjkuje, zato uporabljajo zračno hlajenje, za ceno nekaj učinkovitosti, da paro po izstopu iz turbine kondenzirajo nazaj v vodo.

Ta vrsta sončne elektrarne je najcenejša in na njej je najlažje narediti sistem za shranjevanje toplote, saj ima le en vir toplote (vrh stolpa). Elektrarn v tej obliki je trenutno še zelo malo, vendar predstavlja model, ki ga nameravajo države uporabiti v prihodnosti [62, 31, 63]. Učinkovitost pretvorbe je okoli 17 % [32].

Predlagana je bila ponovna uporaba odsluženega območja kopanja, katerega bi preuredili za solarni stolp. Preureditev v terase na več višinah bi omogočila razporeditev zrcal na različne višine. Na dnu takega odkopa se zbira voda, ki bi jo lahko uporabili za hlajenje pare v kondenzacijskem stolpu ali pri predelavi



Slika 2.5: Solarni stolp [42]. Avtor: Beyond Zero Emissions. Po licenci: [33]

v vodikovo gorivo. Namesto gradnje stolpa s sprejemnikom v sredini bi lahko postavili dva sprejemnika kar na terasah [12].

### 2.1.3 Fotovoltaična elektrarna

Fotovoltaično elektrarno sestavljajo fotovoltaični moduli. V modulih nastane enosmerni električni tok. Nanje bi lahko priključili električne naprave, baterije, akumulatorje, ki jim proizvedeni enosmerni električni tok ustreza. Enosmerni tok pretvorimo v izmenični električni tok, da lahko delež energije vračamo v krajevno omrežje ali uporabimo na domačih električnih napravah. Čez dan se energija shranjuje v dodatnih akumulatorjih ali porabi za segrevanje vode. Na zelo velikih elektrarnah na ravnih strehah ali tleh uporabljajo premične pritrdilne nosilce in sistem za sledenje soncu v dveh oseh. Trenutna pretvorbena učinkovitost modulov (na trgu) je od okoli 15 % do okoli 20 % [59, 63].

Prednost tega sistema je predvsem v poceni fotovoltaičnih modulih. V laboratorijih sicer obstajajo prototipni moduli, ki so sposobni pretvoriti v elektriko

Slika 2.6: Fotovoltaična elektrarna [44]. Avtor: pilot Nadine Y. Barclay za letalske sile ZDA. Slika v javni lasti.

preko 40 % svetlobe [73]. Ta vrsta elektrarne je priporočena tudi za gospodinjstva in manjša podjetja, ker ne potrebuje veliko prostora in dodatne opreme. Manjše domače elektrarne so nameščene na nepremične nosilce na padajočih strehah, kjer je ceneje dodati nove module namesto vzdrževati sistem za sledenje soncu.

Da se energija, porabljena za proizvodnjo in montažo fotovoltaičnega sistema, povrne s pretvorbo sončne energije, že obstaja mera, imenovana čas povračila energije (*energy payback time - EPBT*). Ta znaša 3,3 leta za osončenost severne in srednje Evrope. To je tudi najvišja vrednost iz preglednice v viru [37], saj je za večino sveta, ki leži bližje ekvatorju, osončenost večja. Čas povračila je sicer odvisen tudi od vrste fotovoltaičnih celic. Celotna industrija fotovoltaike bo po ocenah v viru [9] začela vračati energijo pred letom 2020.

Fotovoltaične celice gradijo iz strupenih težkih kovin, vendar enako velja za večino električnih naprav. Vključitev organskih materialov v električne naprave je

Slika 2.7: Fotovoltaika na domovih [44]. Avtor: Gray Watson. Po licenci: [36]

trenutno še raziskovana.

### 2.1.4 Stirlingova elektrarna

Stirlingova elektrarna uporablja množico Stirlingovih prestreznikov (*Stirling dish*). Posamezni prestreznik je veliko okroglo vbočeno (konkavno) zrcalo v obliki sprejemne satelitske antene. V zbirni točki vsakega prestreznika se uporablja Stirlingov pogon ali parni pogon.

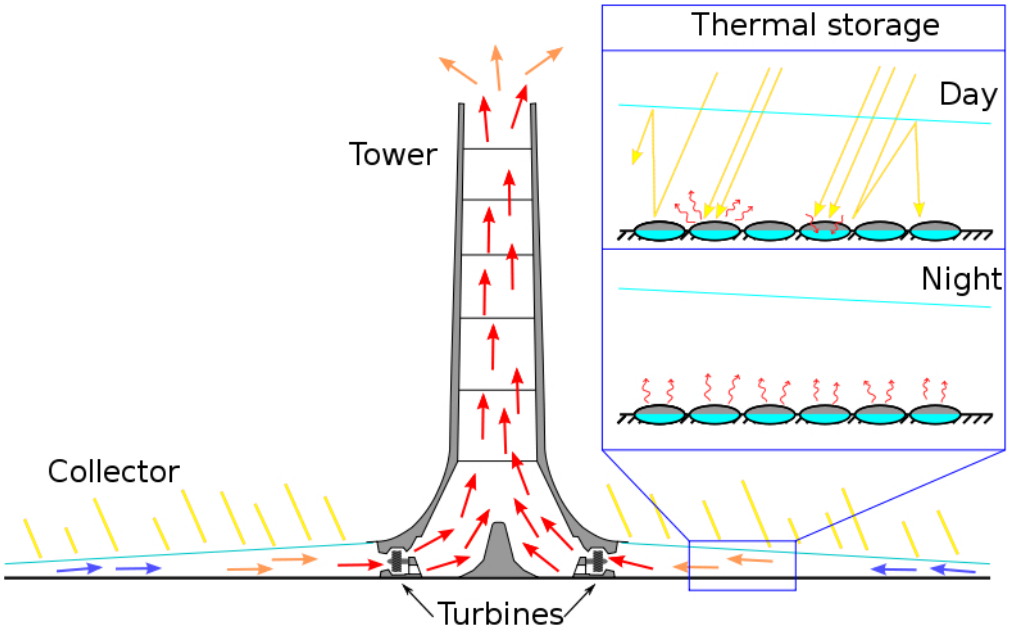
Stirlingov pogon ima znotraj vročo točko, kjer se zbira toplota, in hladno točko. Toplotno razliko med točkama mehanizem s tlačnim batom pretvori v mehansko delo. Vsak prestreznik je samostojna elektrarna. Dosega okoli 30 % učinkovitost v pretvorbi iz sončne svetlobe v električno energijo [68, 57].

To je najboljši sistem pretvorbe energije sonca v električno energijo. Zaradi drage postavitve in vzdrževanja ni razširjen. Večina razvoja poteka v smeri

Slika 2.8: Stirlingova elektrarna [68]. Avtor: Lumos3, angleška Wikipedija. Slika v javni lasti.

Slika 2.9: Malo drugačne oblike Stirlingovih prestreznikov. Levo, avtor: Derek Curry [67]. Desno, avtor: brewbooks [56]. Po licenci: [35].

znižanja stroškov takega sistema, zato se najdejo prestrezniki z raznolikimi oblikami odbojnih zrcal, na primer sestavljeni iz manjših ploščatih.




Slika 2.10: Sončni vetrovni dimnik [64]. Avtor: Kilohn limahn, francoska Wikipedija, za angleško uredil Cryonic07. Po licenci: [36].

### 2.1.5 Sončni vetrovni dimnik

Sončni vetrovni dimnik je velika površina pokrite zemlje, kjer se segreva zrak. Pokritje je nagnjeno, da se segreti zrak pomika proti dimniku v središču z vetrnimi turbinami na vznožju. Torej je na sončno energijo, ki se izkorišča za segrevanje zraka in pretvori v vetrno energijo. Vetrna energija se pretvori v električno. Iz eksperimentov ocenjujejo, da se ne izplača zaradi slabe učinkovitosti pretvorbe v primerjavi s *CSP* ali fotovoltaiko [64].

## 2.2 Vodna energija

Vodna energija je energija premikanja vode. Tok reke, valovanje morske vode ali podvodne tokove ob plimi in oseki pretvarjamo v mehansko delo in nato v električno energijo. Elektrarne na energijo valov ali energijo plime in oseke v Sloveniji niso smiselne, saj je zelo majhna (60 cm razlike med plimo in oseko v Tržaškem zalivu [24]).

### 2.2.1 Zajezitev rek

Vodna elektrarna uporablja za ustvarjanje električne energije vodne turbine. Odvisno od turbin na velikih jezovih dosežejo do 95 % učinkovitost pri pretvorbi vodnega toka v električno energijo. Na manjših jezovih, ki proizvedejo do 5 MW, je učinkovitost od 80 % do 85 % [46]. Da se zagotovi konstantno preskrbo, se zgradi jezove. Z višinsko razliko med mestom izpusta vode in turbinami poleg same količine izpuščene vode nadzorujejo količino proizvedene energije. Opcija je tudi izraba odvečne proizvedene energije iz drugih obnovljivih virov za prečrpavanje nazaj v rezervoar.

Problemi hidroelektrarne so povezani z jezom. Ob gradnji je večkrat potrebno preseliti ljudi in živali višje po toku zajezene reke. V tropskih predelih je potrebno odstraniti tudi rastline. Rastline pod vodo gnijejo in ustvarjajo metan. V rezervoarju lahko zraste mulj in ovira pretok vode. Ker lahko hitro spreminjajo količino izpuščene vode, vplivajo na rečno dno in bregove. Jez lahko poči in povzroči poplave. Upad količine vode predstavlja problem za hidroelektrarne brez jeza. Vseeno je ta oblika bolj varna in ekonomsko smiselna za podjetje, če je v bližini tekoča reka ali potok [47].

### 2.2.2 Izkoriščanje plime in oseke

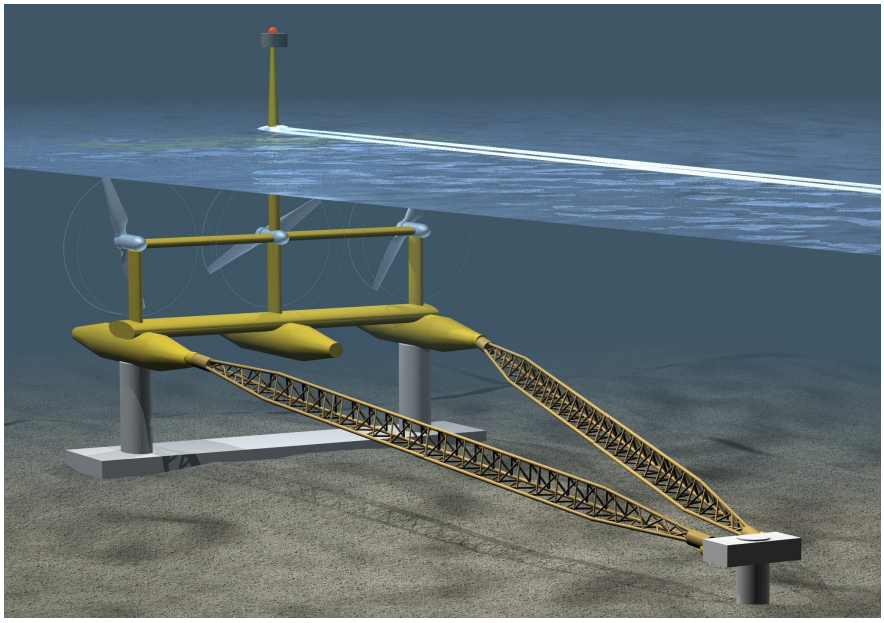
Podvodne elektrarne s turbinami izkoriščajo konstanten vodni tok skozi ožine, ki nastane zaradi plime in oseke ali iz zajezenega rečnega izliva. Rezervoarji, jezovi ali kanali (ožine) so lahko zgrajeni tudi umetno [70].

Problemi so z vplivi na vodne živali in rastline v bližini elektrarne: Stik kemikalij z vodo, elektromagnetno polje, ki ga povzročajo, vibracije, hrup, njihov vpliv na živali, ki se orientirajo z zvokom. Vpliva na razmnoževanje rib, ki za ta namen migrirajo v sladke vode, vendar je bil izliv medtem zajezen. Močan tok lahko ribe potegne v bližino ali pod lopatice turbin. Pritrditev turbin na morsko dno ali zajezitev vpliva na sedimentacijo in spreminja morsko dno.

### 2.2.3 Energija valovanja

Energijo valov lahko koristimo v več oblikah:

- Z dvigovanjem in spuščanjem boje na gladini, ki je pritrjena na morsko dno.



Slika 2.11: Podvodne turbine za izkoriščanje tokov [26]. Avtor: Green Energy Futures - David Dodge. Po licenci: [34].

- S krčenjem in raztezanjem pnevmatskih cevi med členi plavajoče električne kače.

- S pritiskom vala ob zrak, kjer se srečata potopljeni, vstopni del cevi in zračni vrhnji del, potisne zrak proti zoženju na vrhu skozi vetrno turbino. Namestijo jih na obali. Pokončno cev na tem principu lahko postavijo tudi na morju.

- Z zajemom valov v višjeležečih bazenih in sprostitev vode nazaj skozi turbine.

Tudi pri tej obliki so problemi z vplivom elektromagnetnega polja in hrupa mehanizacije na živali. Pri sistemih, pritrjenih na morsko dno, nosilci in podzemni kabel vplivajo na sedimentacijo in spreminjajo morsko dno. Dodatno bi se kjerkoli v mehanizaciji lahko zagozdile živali. In tudi ladje bi se utegnile zaplesti ali trčiti v te sisteme [71].

## 2.3 Vetrna energija

Vetrna energija je energija gibanja zračnih mas. Uporabljamo jo v mlinih na veter. Večja in bolj neprekinjena je v višjih legah ali blizu morja. Idealistična uporaba predstavlja izrabo vetrne energije na velikih višinah s pomočjo balonov, ki bi držali turbino na kablu ali drugih zračnih plovilih. Alternativa je, da bi jih vpeli med gorske vrhove ali visoke zgradbe. Tovrstna postavitev ima veliko problemov. Preko stroškov, zaščite pred strelami, hladnimi temperaturami, nestabilnostjo pozicije do oddaljenosti lokacije za vzdrževanje [45]. Tako so turbine v praksi postavljene v bližnjem hribovju, na morju, blizu morja, na širnih poljih ali znotraj mesta.

Vetrno energijo izkoriščajo z vetrnimi turbinami, ki jih postavijo na visok drog. Lopatice zajamejo vetrno energijo, jo izkoriščajo za mehansko delo in pretvorijo v električno energijo. Na splošno velja, da večje lopatice pomenijo večjo količino zajete energije. Pomembna je oblika lopatic (aerodinamika, za obiskovalce mesta tudi estetika). Velike vetrnice se postavljajo na območjih, kjer veter piha najmanj s hitrostjo 6 metrov na sekundo [69]. Za manjše potrebujemo nad 4 metre na sekundo.

Spopadajo se predvsem s predsodki ljudi, da uničujejo pokrajino, škodijo zdravju živali v bližini, ubijajo ptice. Večina sproženih raziskav je zaključila, da so očitki neutemeljeni. Pri ljudeh so opazili škodo na zdravju zaradi stresa, vendar so ljudje krivili vetrnice. Nazadnje ptice se v normalnih vetrnih pogojih izogibajo vetrnicam. Ptice večinoma umirajo iz drugih razlogov, kot so visokonapetostni kabli in okna visokih zgradb. Za dodatno varnost vetrnic poskrbijo z radarji, ki vetrnice ustavijo, ko so v bližini ptice [72].

## 2.4 Biomasa

Biomasa zajema pridobivanje energije ali proizvodnjo goriva iz lesa, rastlin, živali in organskih odpadkov. Človek izkorišča biomaso, odkar je znal zanetiti ogenj. Sedaj se uporablja za proizvodnjo alkoholov (predvsem etanola) kot primarno biogorivo ali dodatek dizlu [25].

Etanol proizvajamo iz sladkornega trsta in koruze, ki ju gojijo specifično v ta namen. Rastline, ki bodo gojene v namen pridobivanja goriva, potrebujejo ločeno površino. Za biogorivo uporabljamo tudi rastlinska olja ali živalske masti

iz proizvodnje hrane.

Les iz pogozdovanja, lesno-predelovalne industrije ali iz obrezovanja, organski odpadki in ostanki rastlin (stebla, trava) tudi tvorijo biomaso, ki se predvsem sežiga za proizvodnjo plina poleg oglja v termoelektrarnah. Alternativa je kemična predelava lesne in rastlinske biomase v etanol ali vodik in njuna uporaba v ustreznih gorivnih celicah za proizvodnjo električne energije. Izdelava biogoriv iz celuloze rastlin je še vedno v raziskavi. Raziskujejo tudi alge za proizvodnjo biogoriva.

Sežig biomase enako kot fosila goriva povzroča izpuste $CO_2$, ogljikovega monoksida, dušikovega oksida, črni ogljik... Ker je ogljik ujet v rastline in ga s sežigom vračajo v ozračje, je uporaba biomase veljala za ogljično nevtralno. Vendar odvisno od vrste biomase lahko povzroča tudi večje izpuste od fosilnih goriv. Druge vrste uspešno zajemajo ogljik kljub žetvi ali obrezovanju te rastline, saj ponovno zraste. Pri gozdovih se sveže posajeni gozd ne more primerjati s starejšim. Prav tako je problem z zbiranjem, transportom biomase, kar predstavlja stroške in veča izpuste ter vzpodbuja pretirano sečnjo gozda ali druge biomase v bližini elektrarne, zato se biomasa primarno uporablja za predelavo v biogorivo.

## 2.5 Geotermalna energija

Geotermalno energijo koristimo v parnih turbinah na lokacijah, kjer na zemeljsko površje prodre vroča para. Ker takih mest po svetu ni veliko, lahko sami zvrtamo v zemeljsko skorjo parni vrelec in vodnjak za dovod vode. Z napredkom v tehnikah vrtanja jo lahko uporabljamo tudi po svetu, kjer ni geotermalnih vrelcev [43].

Zaradi neučinkovitosti pretvorbe iz toplote v električno energijo, potrebujemo čim toplejšo paro. V nasprotnem primeru moramo vrelec poglobiti. Probleme predstavljajo izpusti $CO_2$, vodikovega sulfida ($H_2S$), metana in amoniaka ($NH_3$). Te lahko zajamemo in vrnemo nazaj v vrtino z dovodom vode kot obliko zajema in shrambe ogljika (*carbon capture and storage*). Prihaja tudi do lokalne toplotne izčrpanosti vrelca ali pomanjkanja dovoda podzemne vode, zato potrebujemo umetno dovajanje vode.

# Poglavje 3

# Zeleno računalništvo

*The Green Computing Book: Tackling Energy Efficiency at Large Scale* [1] nam predstavi probleme visoko zmogljivih računalniških sistemov, kot so superračunalniki in podatkovni centri. Poraba električne energije na nekaterih od teh sistemov lahko celo preseže porabo manjšega mesta. Zmanjševanje velike porabe in hlajenje takšnih sistemov predstavlja izziv. Sedaj se rešuje z izbiro ustreznega hlajenja, kar potrebuje dodatno energijo. Kontrola porabe električne energije na komponentah predstavlja dodatne možnosti.

## 3.1 Varčevanje na nivoju strojne opreme

Trend v izdelavi procesorjev je večanje stopnje integracije z manjšanjem litografije in dodajanje jeder v notranjost procesorjev. Ta pristop v izdelavi se bo nadaljeval tudi v bližnji prihodnosti [7]. Pri tem narašča količina tranzistorjev na prostor znotraj procesorja, s tem tudi količina potrošnikov električne energije in toplota. Problem je torej v gostoti električne moči (*power density*), torej električni moči, ki se troši na omejenem območju procesorja [7].

Da bi znižali porabo in toploto, se izbere nižja stopnja električne napetosti, pri kateri tranzistorji delujejo. Znižanje stopnje električne napetosti omeji frekvenco, pri kateri procesor deluje, in njegovo zmogljivost. Izdelovalci procesorjev lahko nižjo zmogljivost nadomestijo z večjim paralelizmom [7]. Želijo povečati zmogljivost ob enaki porabi električne energije, kot jo ima prejšnji model procesorja. Z dodajanjem jeder povečajo porabo električne energije v procesorju in

izenačijo prihranke električne energije, ki izvirajo iz delovanja pri nižji frekvenci [7]. Uravnavanje stopnje napetosti se lahko uporabi na kateremkoli električnem vezju, ne nujno na centralni procesni enoti. Primeri: grafični procesor, pomnilnik, komunikacijska vodila...

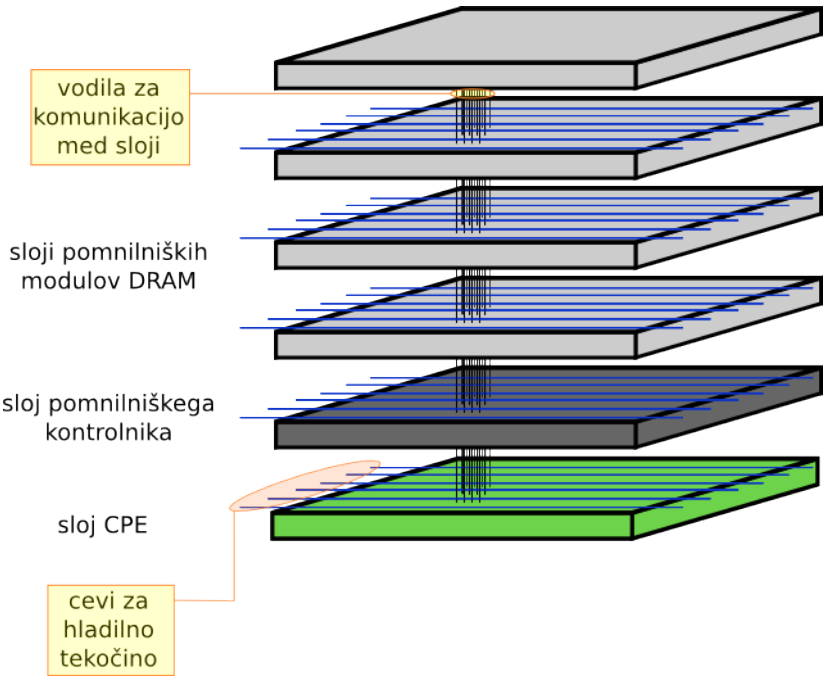
Za izklop odsekov vezja se na njih uporablja "odrez urinega signala" (*clock gating*) ali "odrez napajanja" (*power gating*) [7]. Flip-flopi porabljajo energijo ob preklopih stanja. Uravnavanje frekvence procesorja zato omogoča nižjo porabo energije. "Odrezi" ne omogočajo varčevanja na aktivnem vezju, zato ga je potrebno kombinirati z uravnavanjem napetosti ali frekvence.

Električni šum se zmanjša z znižanjem napetosti na njegovih virih [7]. Toda zniževanje stopnje električne napetosti lahko poveča izgube v dobavi električne energije [7]. Ker procesor spreminja lastno električno napetost, pri kateri deluje, ker komunicira z drugimi napravami, ki imajo spet svoje lastne električne specifikacije, imamo v modernem računalniku eksplozijo različnih stopenj električnih napetosti [7]. Pretvorba na različne stopnje se trenutno izvaja na zunanjem pretvorniku. Iz omenjenih vzrokov, bi lahko nekaj pretvornikov za kontroliranje napetosti v obliki vezja, ki predstavlja preklopni stabilizator (*switched-capacitor circuit*), prenesli v notranjost samega procesorja [7].

Zaradi količine električne energije na vhodno-izhodnih napravah in komunikacijah z njimi bi lahko skrajšali komunikacijske poti s pomočjo tehnik tridimenzionalne integracije procesorja (slika 3.1) [7] in vgradili, na primer, delovni pomnilnik znotraj procesorja. Tridimenzionalno integracijo lahko dosežejo z uporabo izolacijskih premazov in združevanjem rezin vezja (*wafer bonding*) ali preko silicijevih nosilcev (*silicon carrier*) nanašajo nove sloje vezja [7]. Za komunikacije izven procesorja bi lahko uporabili optične povezave [7].

Primere integracije centralne procesne enote in grafičnega procesorja najdemo v seriji procesorjev "A" podjetja AMD. In glede na vir [28] ob polni obremenitvi potrebujejo manj električne energije v primerjavi s samostojnimi centralnimi procesnimi enotami "FX". Serija "FX" potrebuje tudi ločeno grafično kartico, kar pomeni še večjo porabo.

Glede na vir [21] se je korporacija *IBM* lotila izdelave prototipa procesorja po njihovi verziji tridimenzionalne integracije. Zaradi toplote so se jim topile povezave med sloji vezja, kar so rešili s kot las tankimi hladilnimi cevmi med




Slika 3.1: Primer 3D integracije procesorja in delovnega pomnilnika. Narisano po viru [21].

sloji. Torej ta čip lahko hladijo le s tekočino. Spremembe v načinu izdelave in hlajenja procesorja bodo nujnost. Iz vira [7]: "Metode pakiranja vezij se morajo spremeniti, da zmanjšamo induktivnost in električno upornost". Za podrobnosti in z matematičnimi primerjavami podkrepljeno razpravo tehnik, naštetih v tem razdelku, si, prosim, oglejte vir [7].

## 3.2 Dinamično uravnavanje frekvence procesorja

Po specifikaciji *Advanced Configuration and Power Interface - ACPI* je mogoče operacijskemu sistemu razkriti procesorjeve frekvence, med katerimi lahko operacijski sistem preklaplja. Razkritje je potrebno vklopiti v *Basic Input/Output System - BIOS*. Za moj procesor, osem-jedrni AMD FX-8350 se tehnika (in opcija v *BIOS*) imenuje *Cool 'n' Quiet*, na prenosnih napravah *PowerNow!*, na grafičnih

karticah *PowerTune* in *ZeroCore Power*. Intel na svojih procesorjih to imenuje *SpeedStep*, VIA Technologies uporablja *LongHaul (PowerSaver)* [39].

Veliko imen, enak princip delovanja. Spreminjanje frekvence dosežemo preko spreminjanja nivoja električne napetosti na procesorju. Tako s spreminjanjem frekvence hkrati spreminjamo tudi električno napetost. Sistem se imenuje "dinamično skaliranje napetosti in frekvence" (*dynamic voltage and frequency scaling - DVFS*), upočasnjevanje CPE (*CPU throttling*) in kot obratni izraz navijanju (*overclocking*) procesorjev, "podvijanje" (*underclocking*).

Ker uporabljam Kubuntu 14.04 *Long Term Support*, lahko opišem in potrdim delovanje le na tem operacijskem sistemu. Jedro Linuxa je začelo podpirati ACPI od verzije 2.6 naprej [22]. Vsako jedro ima svoj imenik z nastavitvenimi datotekami in je torej obravnavano kot samostojen procesor. Na uradnih Ubuntujevih distribucijah imamo nastavitve za frekvence v imenikih

`/sys/devices/system/cpu/cpu*/cpufreq/.`

'*' stoji za identifikacijsko številko posameznega jedra (0 do 7 v mojem primeru za 8 jeder). *CpuFreq* je zgolj naziv sistema, ki se uporablja za nadzor frekvenc na operacijskih sistemih Linux. Če v *BIOS Cool 'n' Quiet* ne vklopimo, mape cpufreq za posamezno jedro izginejo in ostanemo brez možnosti nastavitev frekvenc.

Če želimo izvedeti trenutno frekvenco, lahko uporabimo ukaz za izpis tekstovne datoteke, kjer operacijski sistem Linux hrani informacije o procesorjih in filtriramo le informacije o frekvencah posameznih jeder:

```
cat /proc/cpuinfo | grep MHz
```

Nastavljanje frekvence je mogoče preko imen profilov spreminjanja frekvence ("guvernerja" nad frekvencami) ali izbiro frekvenc, ki so na voljo. V imeniku posameznega procesorja boste našli datoteke:

- scaling_available_frequencies z naštetimi možnimi frekvencami v kHz, odvisnimi od procesorja, ki ga posedujete. Vsebina: 4000000 3400000 2800000 2100000 1400000
- scaling_available_governors z naštetimi stili spreminjanja frekvenc v obliki imen profilov. Vsebina: conservative ondemand userspace powersave perfor-

mance. *Conservative* in *ondemand* preklapljata po območju možnih frekvenc glede na obremenjenost procesorja. *Conservative* je le počasnejši, bolj zadržan pri tem.

- scaling_cur_freq sporoči trenutno frekvenco (v kHz). Za razliko od /proc/cpuinfo imamo samo trenutno uporabljano frekvenco brez navedene enote (kHz) in brez ostalih informacij o procesorju.

- scaling_driver pove ime gonilnika, ki je uporabljen za kontrolo. Privzeto je to acpi-cpufreq. Torej *BIOS* skozi *ACPI* preda seznam frekvenc operacijskemu sistemu Linux. Starejši procesorji potrebujejo njim namenjen gonilnik.

- scaling_governor je nastavitvena datoteka, kjer se uporabi vpisano ime "guvernerja". Privzeta vsebina: ondemand.

- scaling_max_freq za maksimalno dovoljeno frekvenco od možnih (v kHz). *Performance* profil takoj nastavi frekvenco na vpisano v tej datoteki in je ne znižuje.

- scaling_min_freq za minimalno dovoljeno frekvenco od možnih (v kHz). *Powersave* profil takoj nastavi frekvenco, vpisano v tej datoteki, in je ne viša.

- scaling_setspeed za ročno nastavljanje frekvence, predpisane od administratorja. Upošteva se le pod *userspace* profilom. Vsebina: <unsupported>. Pravzaprav lahko nastavimo frekvenco kar preko okleščenja preklopnega območja na eno frekvenco v datotekah za minimalno in maksimalno frekvenco. To je tudi način, ki sem ga uporabil.

Privzeto se spreminja frekvenca (med 1,4 do 4 GHz) glede na delovno breme (*on demand*). Če beremo samo dokumente in brskamo po spletu, ne moremo polno obremeniti 4 GHz procesorja, zato njegova frekvenca ostaja pri 1,4 GHz. Ob bolj zahtevnih aplikacijah operacijski sistemi sami zvišujejo frekvenco in jo znižajo, ko se zahtevne aplikacije končajo. Zaradi segrevanja jedra je breme aplikacije izmenjano med jedri čez nekaj časa, da se toplota porazdeli po območju procesorja. Večina naprav uporablja tak pristop.

Poskusno sem poganjal poenostavljen algoritem N-teles z 10000 telesi postavljenimi na deset enot razdalje okoli koordinatnega izhodišča. Vsa telesa imajo

enako maso ene enote. V 1000 korakih preračuna medsebojne vplive in premika telesa. Algoritem N-teles je katerikoli algoritem, ki računa medsebojne vplive za večje število teles. Na primer medsebojne vplive gravitacije v množici planetov. Pri tem se problem razbije na računanje vplivov le med paroma teles. Uporabil sem lastno prilagojeno kodo iz naloge za predmet Vzporedni in porazdeljeni sistemi in algoritmi.

Zaporedna verzija algoritma je potrebovala 2530 sekund. Pri 120 vatih je produkt s časom 303600 vat-sekund. Pretvorba v vzporedno kodo za osem jeder je bila izvedena s pomočjo OpenMP. Bazna poraba ob nizki aktivnosti procesorja, preden poženem algoritem, je 77 vatov. Iskali bomo najvarnejšo izvedbo algoritma N-teles.

| stanje po *ACPI* | frekvenca [GHz] | električna moč [W] | električna napetost [V] [23] | čas izvajanja [s] | produkt porabe in časa [Ws] |
|---|---|---|---|---|---|
| P0 | 1,4 | 101 | 0,875 | 1326 | 133926 |
| P1 | 2,1 | 115 | 1 | 878 | 100970 |
| P2 | 2,8 | 133 | 1,125 | 665 | 88445 |
| P3 | 3,4 | 153 | 1,225 | 546 | 83538 |
| P4 | 4,0\|4,1\|4,2 | 196 | 1,344\|1,4\|1,425 | 461 | 90356 |

Tabela 3.1: Tabela rezultatov meritev za algoritem n-teles.

*Turbo CORE* tehnologija (oziroma *Turbo Boost* pri Intelu) preda procesorju odločitev za rahlo navitje frekvence čez najvišjo na izbiro. V mojem primeru lahko s 4 GHz navije na 4,1 Ghz ali do 4,2 GHz. Cpuinfo datoteka višjih frekvenc od teh, ki so na izbiro, nikoli ne izpiše [53]. V datoteki

```
/sys/devices/system/cpu/cpufreq/boost
```

imate zgolj 0 ali 1, ki le odloči, ali procesor lahko uporabi (1) tehniko začasnega navitja ali ne (0). V mojem primeru sem to opcijo pustil omogočeno, njen vpliv na rezultate meritev v tabeli 3.1 ni znan. Procesor odloča, do katere frekvence in kdaj se bo navitje zgodilo [53].

V rezultatih v tabeli 3.1 opazimo korist v uporabi vzporedne oblike. Glede porabe se res najbolje izkaže najnižja frekvenca, toda podaljšanje časa izvajanja pokaže, da to niso najbolj varčne nastavitve procesorja za algoritem N-teles. Pri 3,4 GHz, s časom 546 sekund in 83538 vat-sekund totalne porabe so najbolj varčne nastavitve za moj procesor in uporabljeni algoritem N-teles. Poudarjam, da so najbolj varčne nastavitve odvisne od procesorja in tudi od aplikacije.

Medtem ko rezultati dokazujejo, da je mogoče uravnavati porabo električne energije z (ročnim) uravnavanjem frekvence procesorja, to odpre kompleksno vprašanje o iskanju najbolj varčnih nastavitev za izvedbo izbrane aplikacije. V mojem primeru sem aktivno uporabljal le procesor. V superračunalnikih so aktivno uporabljani tudi diski, sistem komunikacije med procesorji in morda grafične kartice ali ko-procesorji. Ko se v iskanje vmešajo mehanizmi za varčevanje vhodno-izhodnih naprav, preprostega odgovora ni.

Prenosne naprave varčujejo z energijo zaradi baterije. Mehanizem uravnavanja frekvence se uporablja tudi na osebnih računalnikih. Ti sistemi so odvisni od interakcije z uporabnikom, da ustvari in umakne delovno breme.

Pri superračunalnikih ni baterije in ni interaktivne manipulacije s programi. Na velikih sistemih se znanstvena aplikacija izvaja nekaj ur na rezervirani množici procesorjev. V času izvajanja je izkoristek procesorjev ves čas večji od 90 % [7]. Uravnavanje po obremenitvi procesorja (guverner “ondemand”) na superračunalnikih ne omogoča varčevanja. Zato je na superračunalnikih za varčevanje z električno energijo potrebno ročno nastavljanje frekvence in sicer z vpisom izbrane frekvence v nastavitvene datoteke. V naslednjih dveh razdelkih si bomo ogledali drugačna pristopa k uravnavanju frekvence za superračunalnike.

## 3.3 Nadzorovano izvajanje za znižanje porabe

Pri interaktivnih aplikacijah je *DVFS* z avtomatično izbiro glede na obremenitev procesorja (v procentih) uporaben postopek zniževanja porabe. Zahteva interakcijo uporabnika in glede na breme njegovih pognanih aplikacij prilagaja frekvenco procesorja z namenom zmanjšanja obremenjenosti baterije [5]. Pri visoko zmogljivih sistemih, kot so superračunalniki in podatkovni centri, kjer je izvajanju ene znanstvene aplikacije dodeljena rezervirana skupina procesorjev, ki imajo ce-

loten čas izkoristke nad 90 %, tak pristop ni smiseln. Na superračunalnikih želimo znižati porabo električne energije in olajšati hlajenje takšnih sistemov [5]. Pravilo dobre prakse pravi, da na vsakih 10°C povišanja temperature se količina okvar na sistemu podvoji [5].

Trenutno se frekvenca preko *DVFS* na visoko zmogljivih sistemih izbira statično, s praktičnimi poskusi (*polling*). Vsako možno frekvenco preizkusijo. Ko poznamo delovanje aplikacije pod vsemi frekvencami, glede na sprejemljive zakasnitve izvajanja izberejo ustrezno frekvenco pred vsakim zagonom aplikacije [5]. Za dinamično uravnavanje vir [5] predstavi algoritem Beta-prilagajanje, ki ta proces izbire frekvence avtomatizira za super-računalniške sisteme.

### 3.3.1 Beta-prilagajanje

Algoritem na vhodu potrebuje seznam nastavljivih frekvenc. Dodatni vhod sta dolžina časovnega intervala za izvajanje kontrole (vnaprej nastavljena na 1 sekundo) in delež dovoljene zakasnitve programa $\delta$ (vnaprej nastavljen na 5 % dovoljeno upočasnitev) v primerjavi z izvajanjem pod najvišjo frekvenco [5]. Pod vsako nastavljivo frekvenco potrebujemo hitrost v milijonih izvedenih ukazov na sekundo - *MIPS* za model predlagan v viru [5].

Kontrola temelji na trditvi, da velja enačba v vsakem kontrolnem intervalu:

$$\frac{T(f)}{T(f_{max})} \approx \frac{MIPS(f_{max})}{MIPS(f)} \approx \beta \left( \frac{f_{max}}{f} - 1 \right) + 1 \tag{3.3.1}$$

V vsakem kontrolnem intervalu naj bi bilo razmerje med časom izvajanja pod izbrano frekvenco $T(f)$ proti času izvajanja pod maksimalno frekvenco $T(f_{max})$, torej je zakasnitev približno enaka razmerju med *MIPS* pri maksimalni frekvenci $MIPS(f_{max})$ in *MIPS* pri izbrani frekvenci $MIPS(f)$. To približno sledi funkciji skrajno desno v enačbi. Ta opiše zakasnitev s pomočjo spremenljivke $\beta$. Če bi bila $\beta = 1$, bi se čas izvedbe podvojil ob polovici maksimalne frekvence. Za $\beta = 0$ se čas ne spreminja [5]. $\beta$ je torej odvisna od konkretnega sistema, na katerem teče algoritem.

Da je znana zakasnitev v trenutnem kontrolnem intervalu, se $\beta$ preračuna glede

na nazadnje izmerjene *MIPS* pod vsako od $n$ nastavljivih frekvenc [5].

$$\beta = \frac{\sum_{i=1}^{n} \left(\frac{f_{max}}{f_i} - 1\right) \left(\frac{MIPS(f_{max})}{MIPS(f_i)} - 1\right)}{\sum_{i=1}^{n} \left(\frac{f_{max}}{f_i} - 1\right)^2} \tag{3.3.2}$$

Ko imamo $\beta$ izračunano za nazadnje izmerjene *MIPS*, iščemo frekvenco za trenutni interval, da zakasnitev za trenutni interval ne bo presegala zakasnitve, dovoljene od uporabnika $\delta$.

$$f_{predlagana} = max\left(f_{min}, \frac{f_{max}}{1 + \frac{\delta}{\beta}}\right) \tag{3.3.3}$$

Na začetku algoritem za trenutni interval uporabi še nepreizkušene nastavljive frekvence in si zapomni izmerjene *MIPS*. Ko imamo *MIPS* izmerjen za vse nastavljive frekvence, se po preteku kontrolne periode (privzeto vsako sekundo) zgodi zanka [5]:

1. Izračuna se aktualni $\beta$ po formuli 3.3.2.
2. Nato na podlagi $\beta$ izračunamo predlagano frekvenco $f_{predlagana}$ po formuli 3.3.3.
3. Pretvorimo predlagano frekvenco v dejansko nastavljivo frekvenco in jo za trenutni interval uporabimo.
4. Posodobimo *MIPS* statistiko za izbrano frekvenco, ko se kontrolni interval konča. Sklepam, da bi izmerjene *MIPS* lahko prebrali in posodabljali na začetku kontrolnega intervala, vendar v viru [5] to ni navedeno.

Algoritem je na testni skupini procesorjev Athlon64 dosegel povprečno 12 % znižanja porabe energije in povprečno 4 % zakasnitve izvajanja [5]. Na skupini procesorjev Opteron prihranimo povprečno 18 % ob povprečno 5 % zakasnitvi [5]. Torej je uspešno prepoznaval trenutke v NAS-MPI znanstvenih aplikacijah, ki so bile manj zahtevne do procesorja in so se osredotočale na komunikacijo izven procesorja [5].

Predlagana frekvenca ima vrednost med strojno podprtima frekvencama, ki ju lahko izberemo. V viru [5] predlagajo, da bi del izbranega časovnega intervala izvajali program pod nižjo strojno podprto frekvenco in preostanek pod višjo.

Opcija je, da preprosto popustimo in izvajamo pod najbližjo nižjo strojno podprto frekvenco od predlagane za celoten časovni interval.

Algoritem je mišljen kot časovno sprožen program, saj operacijski sistemi že omogočajo funkcionalnost, kot je na primer urin alarm, ki je dejansko omenjen v viru [5]. Glede na dejstvo, da je namenjen za superračunalnike, kjer množica procesorjev izvaja le eno aplikacijo, navedba aplikacije kot vhoda v algoritem ni potrebna. Poganjati bi ga morali na vsakem jedru posebej [5] kot sem že opozoril v razdelku 3.2, ker Linux vidi vsako jedro kot samostojen procesor.

## 3.4 Strojno učenje za iskanje varčne porabe

Pri vseh vzporednih aplikacijah srečamo omejitve v prilagodljivosti programa na poljubno število procesorjev, jeder ali strojno podprtih niti. Vzroki za te omejitve so lahko v zasnovi programa [2]:

- Preprosto je bil zasnovan in tudi programiran za, na primer, dve niti.
- Za pravilno delovanje potrebuje med-nitno usklajevanje. Na primer preko pregrad (*barriers*), ključavnic (*mutexes/locks*), semaforjev, atomarnih operacij...
- In tudi zaradi strojnih omejitev. Na primer količina jeder, pomnilnika, hitrosti vodila do vhodno-izhodnih naprav...

Zaradi obstoja teh omejitev ni smiselno izvajanje aplikacij z vsemi procesorji in jedri, delujočimi pri maksimalni frekvenci. Zato ob *DVFS* vir [2] predlaga dodatno uporabo sistema, ki bo omejil število aktivnih procesorjev in jeder ter porazdelil niti med aktivnimi jedri. Ta se imenuje *Dynamic concurrency throttling - DCT*. Sistem seveda ni uporaben, če je inicializacija programa odvisna od števila uporabljenih niti [2]. To je ena izmed nastavitev, ki jih spreminjamo. V takih primerih mora število niti ostati konstanta v aplikaciji.

Na večjih računalniških sistemih obstaja veliko število možnih nastavitev števila aktivnih jeder in frekvenc, na katerih lahko izvajamo aplikacijo. Trenutno te nastavitve preiskujejo ročno s preizkusi pod vsemi nastavitvami za posamezno aplikacijo, kot je bilo omenjeno v prejšnjem razdelku. Z vse večjim številom jeder in nastavljivimi frekvencami v procesorjih postaja število kombinacij teh nastavitev vse

večje. Frekvence bomo predstavili s spremenljivko 'L', za jedra pa uporabili 'C'. Število kombinacije vseh nastavitev tako narašča po funkciji $O(L*2^C)$ [2]. Vendar s tem odkrijejo nastavitve, ki so namenjene točno določeni, celotni aplikaciji.

Da je zadeva še bolj komplicirana, imajo aplikacije faze izvajanja [2]. Primeri faz so: priprava pomnilniškega prostora, branje iz datoteke, prvi del postopka, drugi del postopka..., pisanje rezultatov v novo datoteko. Na primer faza branja in pisanja na disk predstavlja priložnost za znižanje frekvence. Zato za faze želimo sprožiti iskanje najbolj ustreznih nastavitev.

Vir [2] namesto preizkusov uporabi strojno učenje s prilagojenimi znanstvenimi aplikacijami, ki uporabljajo OpenMP (*Open Multi-Processing*). Da bi aplikacije lahko uporabljale strojno učenje, so predstavili *framework*, imenovan *Adaptive Concurrency Throttling Optimization Run-time system - ACTOR*, ki ga morajo aplikacije uporabiti za kontrolo in povezavo s prediktorjem.

Prediktor temelji na linearni regresiji in deluje v času izvajanja programa [2], da lahko predvidi najbolj ugodne nastavitve med izvajanjem faz aplikacije. Aplikacije znotraj faz izvedejo več iteracij. To lastnost so v viru [2] uporabili. Znotraj posamezne faze *ACTOR* požene nekaj začetnih iteracij pod omejeno množico nastavitev. Rezultate preda prediktorju. Prediktor za vse ostale nastavitve predvideva rezultate in *ACTOR* nastavi predvidene najugodnejše nastavitve. Preostale iteracije faze se izvedejo pod izbranimi nastavitvami. Ta pristop ni uporaben, če ima ena izmed iteracij zelo drugačno delovno breme [2], na primer zaradi vejitve kode na podlagi identitete niti.

### 3.4.1 Linearni regresor

Prediktor kot vhod sprejme poljubno število strojnih dogodkov in uporabnih ukazov na cikel - *uIPC*. Z "uporabnih" ukazov je mišljeno, da ne štejejo ukazov, namenjenih paralelizaciji, kot so ustvarjanje niti, med-nitna kontrola... [2] Strojni dogodki so dostop do prvega nivoja predpomnilnika, dostopi do delovnega pomnilnika, diska... uIPC so bili merjeni pod nastavitvami, ki jih označimo s spremenljivko 'S', torej $uIPC_S$. Predvidevamo obnašanje faze aplikacije, IPC na ciljnih nastavitvah, ki jih označimo s spremenljivko 't', torej $uIPC_t$ [2].

V praksi so strojni dogodki, ki najbolj vplivajo na IPC, odvisni od sistema, na katerem deluje prediktor. In število dogodkov, ki jih lahko spremljamo med

izvajanjem, je strojno omejeno s številom specializiranih, namenskih registrov [2]. Na testnem sistemu v viru [2] so imeli dva 4-jedrna procesorja Intel Xeon E5320 z dvema frekvencama na izbiro: 1,6 in 1,86 GHz.

Ta procesorja sta imela dva namenska registra. Prvega so morali nameniti štetju $IPC_S$. Drugega so namenili štetju dostopov do prvega nivoja podatkovnega predpomnilnika [2]. Ta strojni dogodek je bil najbolje koreliran z IPC. Strojne dogodke, ki so najprimernejših za aktivno spremljanje, lahko avtomatsko izberemo s sortiranjem po korelaciji in omejimo glede na število namenskih registrov [2].

Pri učenju prediktorja uporabimo znanje o računalnikovi arhitekturi in za učne vzorce izberemo nastavitve, ki spreminjajo frekvenco (preko *DVFS*) in število niti z njihovo topologijo (preko *DCT* z uporabo ukaza sched_setaffinity() in ukaza omp_set_num_threads() iz *OpenMP* knjižice). Pri tem izberemo nekaj možnih in izpustimo simetrične [2]. Primer: [(1 procesor, 2 niti z nedeljenim L2 predpomnilnikom, 1. frekvenca), (2 procesorja, 2 niti v vsakem z deljenim L2 predpomnilnikom, 2. frekvenca), (2 procesorja, 4 niti, 2. frekvenca)]. Med njimi mora biti nastavitev, kjer poženemo vsak procesor z maksimalnim številom niti pri maksimalni frekvenci [2].

Sedaj si oglejmo opisano z matematičnega pogleda za $|S|$ vzorcev:

$$uIPC_t = \sum_{i=1}^{|S|}(uIPC_i * \alpha_{(t,i)}(e_{(1...n,i)})) + \lambda_t(e_{(1...n,S)}) + \epsilon_t \tag{3.4.1}$$

$\alpha_{t,i}$ je funkcija nad pogostostmi (frekvencami) strojnih dogodkov (*event rates*) 'e', katerih imamo od 1 do 'n'. $\lambda_t$ enako, $\epsilon_t$ predstavlja ostanek v linearni funkciji pri pretvarjanju nastavitev vzorcev v ciljne nastavitve [2].

$$\alpha_{(t,i)}(e_{(1...n,i)}) = \sum_{a=1}^{n}(x_{(t,a,i)} * e_{(a,i)} + \gamma_{(t,a,i)}) + z_{(t,i)} \tag{3.4.2}$$

$\alpha$ je funkcija nad frekvencami dogodkov 'e'. Predstavlja preučevanje vplivov posameznega dogodka, indeksiranega z 'a' iz 'i'-tega vzorca, s svojimi nastavitvami, nad ciljnimi nastavitvami 't'. S povečanjem/manjšanjem imitiramo frekvence dogodkov na ciljnih nastavitvah. Koeficiente (x) in ostanke ($\gamma$, $z$) določi linearni regresor [2].

$$\lambda_t(e_{(1...n,S)}) = \sum_{a=1}^{n}(\sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,a,j,k)} * e_{(a,j)} * e_{(a,k)})))+ \sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,j,k,IPC)} * uIPC_j * uIPC_k)) + \iota_t \quad (3.4.3)$$

$\lambda$ je funkcija nad medsebojnimi vplivi med uporabljenimi vzorci [2]. Prvi odsek preučuje vpliv med frekvencama istih dogodkov 'e', indeksiranima z 'a' izmed različnih vzorcev 'j' in 'k'. Drugi odsek preučuje vpliv med uIPC iz različnih vzorcev. Zato mora linearni regresor določiti koeficiente posebej med dogodkoma ($\mu_{(t,a,j,k)}$) in ločeno še med uIPC ($\mu_{(t,j,k,IPC)}$) ter konstanto $\iota_t$.

### 3.4.2 Rezultati strojnega učenja

Po izvedbi testov so v viru [2] ugotovili, da so programi občutljivi na topologijo niti. Torej kateremu jedru so dodeljene. Medtem ko so performanse praktično enake, lahko nastanejo velike razlike v potrošnji električne energije. Niti redko uspejo koristiti prednosti deljenega drugega nivoja predpomnilnika, saj pride do tekme zanj (*memory contention*) [2]. Tako ima prednost porazdeliti dve niti v ločena modula s privatnim drugim nivojem predpomnilnika.

Predikcija zgolj topologije niti po *DCT* modelu prediktorja omogoča izboljšave performans in prihranke električne energije samostojno. Bolje deluje hibridni model *DCT+DVFS*, saj predvidi delovanje s spreminjanjem števila aktivnih jeder in frekvence. Samostojna uporaba le *DVFS* modela prediktorja ne izboljša performans [2].

Ker bodo procesorji v prihodnosti imeli več jeder in tudi več frekvenc na izbiro ter namenskih registrov za spremljanje strojnih dogodkov, se pričakujejo izboljšave natančnost (hibridnega DCT+DVFS) prediktorja. Ta je bila velika že na tem sistemu, saj je napačno izbral eno izmed 10 najslabših konfiguracij le v 7 % primerov [2].

## 3.5 Usklajevanje porabe strežnikov

Na strežnikih porabo električne energije kontroliramo na nivoju strežnika, omarice, celotnega podatkovnega centra ali celotne geografsko porazdeljene infrastrukture [4]. Rešitve so že implementirane na nivojih strojne opreme, tovarniškega programja (*firmware*), gonilnikov, operacijskega sistema in aplikacije [4]. Za nadzorovanje stanja celotnega strežnika obstaja odprti standard *Advanced configuration and power interface (ACPI)*, ki se izpopolnjuje.

V tej pestrosti mehanizmov pogrešamo koordinacijo, sodelovanje med temi komponentami in mehanizmi, kar pripelje do problemov [4].

- Če preko *DVFS* znižamo frekvenco strežnikovega procesorja, se spremeni sposobnost vseh aplikacij, ki jih gostimo na strežniku. V tem pogledu se na električno potrošnjo gleda kot na vir, ki je na voljo aplikaciji.
- Potrebujemo smiseln mehanizem razvrščanja delovnega bremena med strežniki, da nobenega ne preobremenimo (zasujemo z delovnim bremenom). Nekateri strežniki utegnejo biti izklopljeni ali njihova storilnost znižana zaradi trenutnih nastavitev frekvence.
- Pri upravljanju potrošnje imamo različne cilje.
  - Sledenje (*to track*) časovnim rokom, pri čemer nadzornik poišče minimalno porabo, da se aplikacija izvede do roka.
  - Omejevanje (*to cap*) porabe, da nimamo termalnih deformacij na komponentah, da ne prekoračimo mesečnega proračuna. Pri tem žrtvujemo storilnost strežnika.
  - Optimizacija (*to optimize*) je kompromis med potrošnjo energije, odzivnostjo, hlajenjem... Tako časovni rok, omejitve porabe kot tudi temperature komponent so uporabljene v funkciji, na podlagi katere se izračunajo nove kompromisne nastavitve za celoten strežnik.

Ob tej raznolikosti mehanizmov, ki so ločeni, samostojni, izolirani, z različnimi nameni uporabe, prihaja do konfliktov in vmešavanja med seboj. Primer je konflikt med mehanizmoma omejevanja in sledenja, kar ima za rezultat lahko prekoračitev proračuna ali slabe zmogljivosti ali odpoved sistema zaradi pregretja [4].

Neusklajenost med mehanizmom omejevanja in distribucijo delovnega bremena bi povzročila preobremenitev enih in neizkoriščenost drugih strežnikov. Neizkoriščeni strežniki bi lahko postali kandidati za izklop. Delo bi se zgostilo na že polno obremenjene, ki delujejo pri nižji frekvenci zaradi omejitev v porabi energije, v času pojavitve (*burst*) novih zahtev. Skratka nestabilno delovanje celotnega podatkovnega centra [4].

### 3.5.1 Hlajenje

Tudi oprema in stroški za opremo niso zanemarljivi. Raziskave so pokazale, da na vsak vat energije potrošimo od pol do celoten dodatni vat za opremo zunaj strežnika [4]. Ta vključuje ventilatorje, hladilnike, klimo računalnike sobe (*computer room air conditioner - CRAC*), napajalnike (*power delivery/supply units - PDU/PSU*), sisteme za brezprekinitveno napajanje (*uninterruptible power supply - UPS*) ali v primeru vodnega hlajenja: hladilne stolpe, črpalke [4].

Problem je kompleksen v takih sistemih, ker imamo znotraj strežnika več mehanizmov za kontrolo porabe, več virov informacij in potrebo po kontroli nad napravami tako znotraj kot zunaj strežnika. Med informacijami na primer temperatura, hitrost pretoka zraka/vode in v primeru vode tudi pritisk, da vemo če je primanjkuje. Pri tem ne smemo prekoračiti temperaturnega limita niti na strežniku niti v večjem obsegu.

### 3.5.2 Osnutek sistema koordinacije

Problem koordinacije je eden izmed problemov optimizacije s pogoji [4]. Iščemo minimalno porabo za računalniški sistem in za hladilni sistem sobe, v kateri je računalniški sistem postavljen. Električni moči 'P', ki ju porabljata računalniški sistem in hladilni sistem zunaj strežnikov, sta odvisni od časa 't' in '$\tau$' [4]. Razlogi: obdobje nizke porabe, obdobje visoke porabe, pametna omrežja - *smart grid*, aktivno spreminjanje porabe preko *DVFS*, vklop/izklop nekaterih strežnikov:

$$min \int_0^t (P_{streznikov}(\tau) + P_{hladilnegasistema}(\tau)) * d\tau \tag{3.5.1}$$

Pri tem ne smemo preseči virov (*server utilization*) na ravni strežnika ali celega podatkovnega centra [4]. V kontekstu tega razdelka je ta vir obremenjenost

procesorja:

$$Utilizacija_{streznika} \leq Utilizacija_{meja} \tag{3.5.2}$$

ali temperaturnih 'T' omejitev katerekoli komponente, celotnega strežnika ali na večji ravni [4]:

$$T_{streznika} \leq T_{meja} \tag{3.5.3}$$

In ne smemo preseči proračuna električne energije, ki se lahko v vsakem trenutku spremeni. To je količina energije v vatih [4]:

$$P_{streznikov}(\tau) + P_{hladilnegasistema}(\tau) \leq P_{proracuna}(\tau) \tag{3.5.4}$$

Povezava med delovnim bremenom, ki se kaže v obremenjenosti procesorja in električne moči, je pogosto modelirana linearno [4] :

$$P_{streznika}(t) = a_f(t) * obremenjenost_{streznika}(t) + b_f(t) \tag{3.5.5}$$

Vendar se to zgodi le v primeru, da sta parametra ($a_f(t)$ in $b_f(t)$) konstanti.

V praksi električna moč (P) ni v linearni odvisnosti od obremenjenosti strežnikovega procesorja (*server utilization*) [4]. Ko merijo električno moč v odvisnosti od spreminjajočega delovnega bremena, pri vnaprej določeni stopnji *DVFS*, torej fiksni napetosti in frekvenci procesorja, moč narašča linearno. Toda če merimo moč ob spreminjajočih stopnjah *DVFS* ob fiksnem delovnem bremenu, moč narašča hitreje kot linearno [4].

Dokaz je očiten iz formule električne moči kot zmnožek napetosti in električnega toka ($P = UI$) in Ohmovega zakona za električno napetost ($U = RI$). Izrazimo električni tok (I) in ga nadomestimo v enačbi za moč in dobili bomo $P = \frac{U^2}{R}$. Električna moč je kvadratna funkcija električne napetosti [4]. Tudi če spreminjamo le frekvenco (in napetost ostane enaka), se energija porablja ob preklapljanju logičnega stanja v vezju [7]. Nižja frekvenca pomeni manj preklopov logičnih stanj, kar pomeni manjšo potrošnjo.

### 3.5.3 Odzivnost strežnika

Moderne aplikacije so sestavljene iz več (za 'M') stopenj, na primer stopnja povezovanja preko spleta, stopnja aplikacije, stopnja dostopa do podatkovne baze

itd. Vsaka od teh stopenj lahko teče znotraj svojega navideznega stroja (*virtual machine - VM*) [4].

Povprečni odzivni čas (*mean response time - MRT*) celotne aplikacije zahteva prehod skozi vse stopnje in je približno enak vsoti časa, ki ga zahteva od procesorja ($\beta_{nm}$) in časa, porabljenega na drugih tipih virov ($\alpha_n$) kot so disk, omrežje [4]. Časi so dodatno razvrščeni glede na tipe zahtev (za 'N' tipov). '$\lambda_n$' stoji za pogostost prihoda zahtev $n$-tega tipa. Zahteve imajo odmerjeni čas '$E_m$' (*CPU entitlement*), namenjen $m$-ti stopnji aplikacije [4]:

$$MRT \approx \frac{1}{\lambda} \sum_{m=1}^{N} \frac{\sum_{n=1}^{N}(\beta_{nm}\lambda_n)}{E_m - \sum_{n=1}^{N}(\beta_{nm}\lambda_n)} + \sum_{n}^{N}(\alpha_n\lambda_n) \tag{3.5.6}$$

Funkcija opiše odzivni čas na sistemih, kjer uporabljajo migracijo *VM* med fizičnimi strežniki [4]. Uporablja se za preverjanje razporeditve virov strežnikovega procesorja med več *VM* na način, ki ne bo kršil dogovorjene storitvene stopnje. Uporablja se v nadzorniku aplikacije pri arhitekturi s slike 3.4. Primeren je za modeliranje odzivnosti na strežnikih, kjer se zmogljivost njihovih procesorjev aktivno spreminja. Medtem ko stopnja *DVFS* vpliva na storilnost strežnika, čas izvajanja posamezne zahteve ($\beta$) v procesorju morda ne bo spremenjen [4].

### 3.5.4 Temperatura strežnika

Za modeliranje temperature procesorja '$T_{CPU}$' (ali drugih komponent) se lahko uporabi enačba [4]:

$$C_1 \frac{\triangle T_{CPU}}{\triangle t} = \frac{C_2}{R(t)}(T_{prostora} - T_{CPU}(t)) + Q(t) \tag{3.5.7}$$

Levi del enačbe predstavlja temperaturno razliko ($\triangle T_{CPU}$) med dvema meritvama in časom ($\triangle t$) med njima [4]. Odvisna je od prenosa toplote '$Q(t)$' s procesorja v prostor. Toploto lahko približno izračunamo iz električne moči (porabe) procesorja '$P_{CPU}(t)$' [4].

$$P_{CPU}(t) = c_f(t) * Utilizacija_{CPU}(t) + d_f(t) \tag{3.5.8}$$

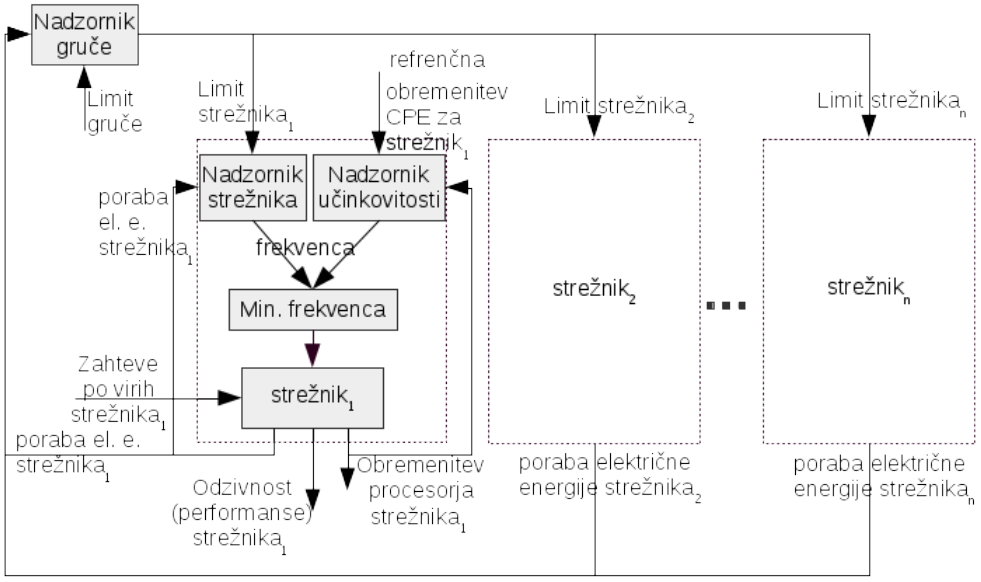
'$R(t)$' predstavlja toplotni upor med senzorjem za procesor in senzorjem temperature prostora, v katerem je strežnik [4]. Približek toplotnega upora ima sledečo

enačbo.

$$R(t) \approx \frac{C_3}{V_j^{n_R}} + C_4 \tag{3.5.9}$$

Prostorninski pretok zraka '$V$' je odvisen od zasnove hladilnika (*heat sink*) [4]. Njegov eksponent ($n_R$) je odvisen od oblike krivulje toplotnega upora kot funkcije intenzivnosti pretoka zraka. In vse štiri konstante ($C_{1...4}$) so povezane s tekočinskimi in materialnimi lastnostmi zraka, pakiranjem procesorja in hladilnikom [4].

### 3.5.5 Arhitekturna porazdelitev nadzornikov



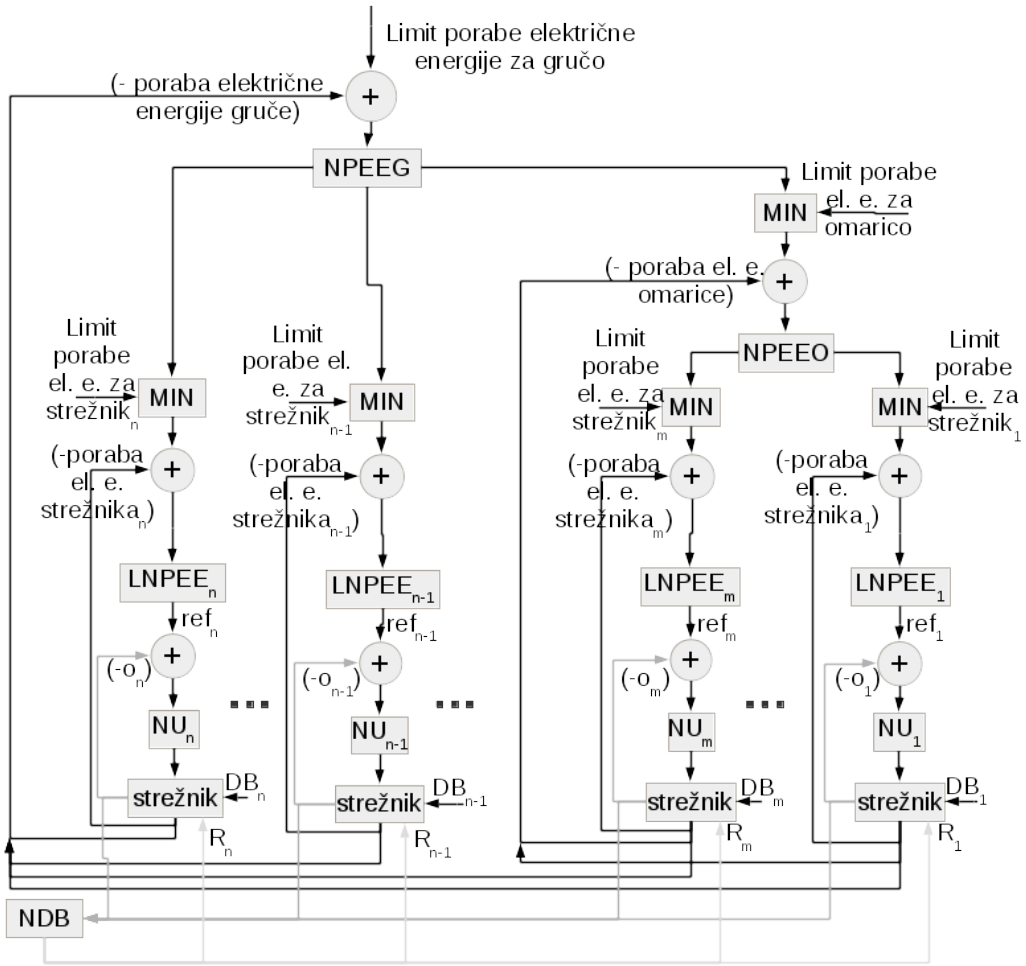

Slika 3.2: Preprosta kontrola nad gručo strežnikov. Narisano po viru [4].

#### Dvostopenjska kontrolna arhitektura

Kot se vidi na sliki 3.2, nadzornik gruče (*Group Capper*) kontrolira gručo strežnikov in upravlja nad električno močjo, ki je na voljo (*power budget*) [4]. Omejeno električno moč dobi nadzornik strežnika (*Server Capper*), ki glede na omejitev spremeni frekvenco procesorju preko *DVFS*. Vsi strežniki sporočajo svojo porabo svojim

nadzornikom (*Server Capper*) in tudi nadzorniku celotne gruče (*Group Capper*) [4].

Če proračun električne energije ni omejen, procesorjevo frekvenco nadzoruje strežnikov nadzornik učinkovitosti (*Efficiency Controller*), ki glede na referenčno vrednost obremenjenosti procesorja določi frekvenco, da obremenitev procesorja sledi referenčni vrednosti. Pri visoki referenčni vrednosti frekvenco zniža in obratno [4]. Strežnik nadzorniku učinkovitosti sporoča dejansko obremenjenost procesorja (*resource utilization*).




Slika 3.3: Nivojna (kaskadno) porazdeljena kontrola nad strežniki. Narisano po viru [4].

### Večstopenjska kontrolna arhitektura

V tej arhitekturi nadzornik porabe električne energije gruče (*group power controler*, NPEEG na sliki 3.3) opravlja isto nalogo, le da porazdeljuje in nadzoruje porabo električne moči (proračun) med posameznimi strežniki in več-strežniškimi omaricami [4]. Lokalni nadzorniki porabe električne energije (*local power controler* LNPEE na sliki 3.3) nato nadzoruje porabo na posamezni napravi, naj je to samostojen strežnik ali vstavljen v omarico (*blade inside enclosure*) [4]. Znotraj omaric nadzornik porabe električne energije za omarico (*eclosure power controler* NPEEO na sliki 3.3) porazdeljuje električno moč med vstavljene strežnike.

Predlagana omejitev (od nadzornikov) in pred-nastavljena se primerjata, na koncu je minimalna izmed njiju uporabljena za nastavitev referenčne obremenitve procesorja ('ref' iz *LPC* na sliki 3.3) [4]. Referenčna obremenitev procesorja je sporočena nadzorniku učinkovitosti (*Efficiency Controller* NU na sliki 3.3). Nazadnje NU preko *DVFS* nastavi ustrezno frekvenco na strežniku [4].

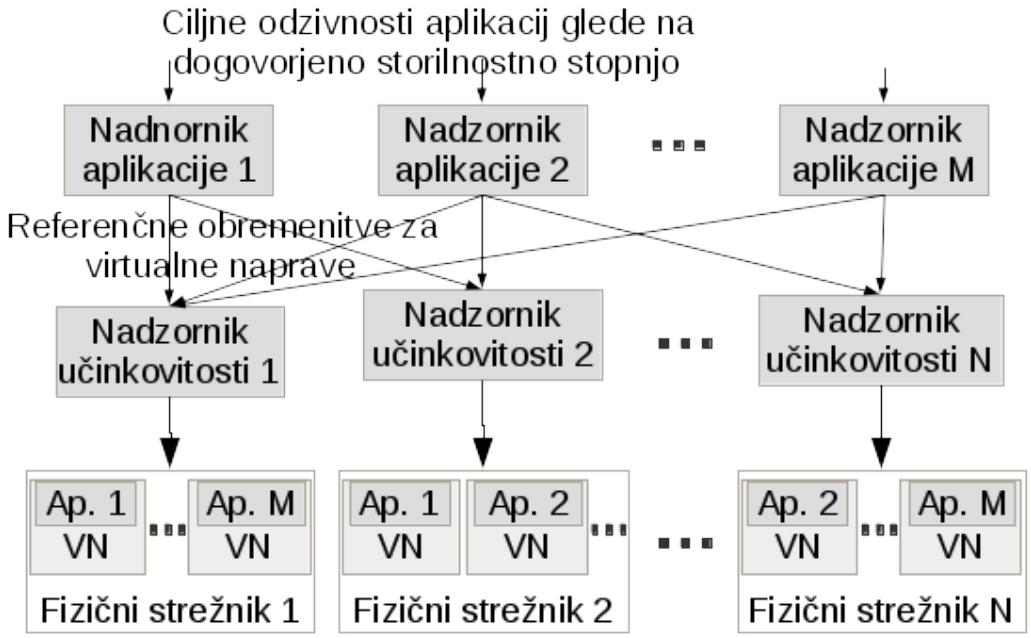
Ločeno imamo v arhitekturi nadzornika delovnega bremena (*Global Controller* NDB na sliki 3.3). Njegova vloga je spremljanje dejanske obremenitve procesorjev ('o' iz strežnikov na sliki 3.3) in porazdeljevanje delovnega bremena (*workload distribution* 'R' na sliki 3.3) med aktivnimi strežniki [4]. NDB lahko ukaže izklop strežnikov in porazdeli delo med preostalimi aktivnimi z "migracijo navidezne naprave" (*VM migration*) in jih po potrebi ponovno zažene [4].

Pri uporabi te arhitekture lahko pride do kršitev pri porabi električne energije zaradi zakasnitev ob prehodu novih vrednosti skozi sistem [4].

### Arhitektura za dogovorjeno storitveno stopnjo

V tej arhitekturi (slika 3.4) so stopnje (*tiers*) aplikacije ločene in se izvajajo znotraj *VM*, porazdeljenih čez več fizičnih strežnikov [4]. Je razširitev arhitektur za kontrolo porabe električne energije. Tokrat nadzorujemo odzivnost aplikacij, da ustrezajo pogojem dogovorjene storitvene stopnje (*service level agreement*). Nadzornik aplikacije (*application controller*) glede na ciljno odzivnost posamezni *VM* predpiše referenčno obremenitev strežnikovega procesorja (*resource utilization*) [4].

Referenčna obremenitev procesorja za *VM* se prilagajo glede na sporočeno (*feedback*) obremenitev strežnikovega procesorja ali na podlagi vnaprejšnje (*feed-forward*) ocene obremenitve s pomočjo formule 3.5.6 in občasnimi popravki ocen




Slika 3.4: Arhitektura za dogovorjeno storitveno stopnjo. Narisano po viru [4].

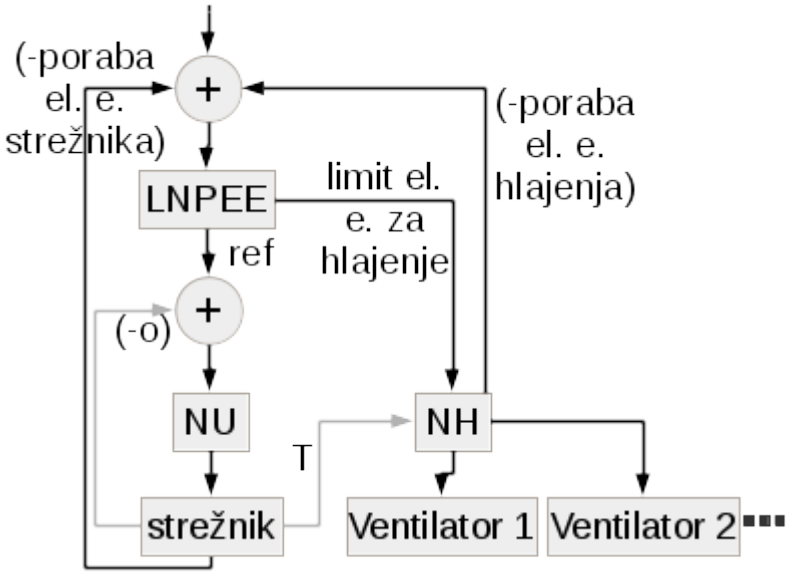
z dejanskimi meritvami delovnega bremena (potrošnjo virov) v realnem času [4].

Nadzornik učinkovitosti (*efficiency controller* na sliki 3.4) skozi *DVFS* nadzoruje porabo električne energije strežnika. Pri tem skuša zadostiti skupnim zahtevam po virih strežnikovega procesorja za vse *VM*, ki tečejo na njem [4]. Dodatno mora razporediti vire (*entitlement for CPU resources*) med *VM*, da so posamezne potrebe po virih zadoščene. Če zahteve presegajo sposobnost strežnika, lahko čas razporedi proporcionalno glede na količino zahtev ali glede na nastavljene prioritete aplikacij [4].

### Vključitev hlajenja

Doslej termodinamike nismo vključili, da bi uporabili formulo 3.5.7. Zgornje arhitekture lahko nadgradimo z nadzornikom hlajenja (*fan controller*) [4]. Lahko bi kombinirali hlajenje z arhitekturo s slike 3.4 s kontrolo nad aplikacijami in arhitekturo s slike 3.3 s kontrolo nad potrošnjo energije strežnikov [4].

Slika 3.5 prikazuje moj primer za več-nivojno kontrolno arhitekturo. LNPEE




Slika 3.5: Hlajenje na večstopenjski kontrolni arhitekturi. Po opisu v viru [4].

porazdeli proračun energije med NU in nadzornikom hlajenja - NH. NH uravnava hitrost ventilatorjev na podlagi temperature strežnikovega procesorja z minimalno porabo energije za hlajenje [4].

### 3.5.6 Delovanje nadzornikov

Ta odsek je zbirka predlaganih algoritmov nadzora nad informacijam v nadzornikih. Z *DVFS* je mogoče frekvenco procesorja nastaviti v nekaj nanosekundah [4]. Kontrolni interval določa odločitev, za koliko časa smemo preseči porabo proračuna električne energije.

#### Nadzornik učinkovitosti

Nadzornik učinkovitosti s slik 3.2 in 3.3 lahko deluje po nadaljnjem opisu. Ta opis je le predlog iz vira [4] in predvideva prilagoditve.

Pri vsakem kontrolnem intervalu (indeksiran z '$j$') se zgodi sledeče [4].

1. Poizvemo povprečno obremenjenost procesorja za prejšnji interval '$util(j-1)$'. Poizvedbo opravimo enkrat in uporabimo pridobljeno vre-

dnost. Podatke lahko preberemo večkrat v preteklem intervalu in uporabimo povprečje pridobljenih vrednosti.

2. Na podlagi obremenjenosti izračunamo frekvenco za trenutni interval:

$$f(j) = f(j-1) - \lambda \frac{f_Q(j-1)util(j-1)}{ref_{util}}(ref_{util} - util(j-1)) \quad (3.5.10)$$

'$f(j-1)$' je izračunana frekvenca iz prejšnjega intervala. '$f_Q(j-1)$' je nastavljena strojno podprta frekvenca iz prejšnjega intervala. '$ref_{util}$' je referenčna obremenjenost procesorja, ki ga nadzorniku učinkovitosti predpiše lokalni nadzornik. '$\lambda$' je konstanta. Nazadnje izračunano frekvenco omejimo v vrednosti:

$$f(j) = min(f_{max}, max(f_{min}, f(j))). \quad (3.5.11)$$

3. Izračunano frekvenco pretvori v ustrezno strojno podprto frekvenco $f_Q(j)$. Primer: $f(j) = 1434,6798 \Rightarrow f_Q(j) = 1400$ glede na moj procesor. Glede nastavljanja frekvenc je bolje razloženo v odseku 3.2.

4. Če procesor ne teče pri strojno podprti frekvenci za trenutni interval $f_Q(j)$, jo nastavi.

Ta proces uporabi povratne informacije od strežnika (*feedback control* [4]), kot smo že omenili v opisih vlog nadzornikov, tokrat le konkretneje predlagamo uporabo teh informacij.

Matematična formula v drugem koraku je zasnovana tako, da se frekvenca hitro odzove na nenaden prihod večje količine zahtev in se počasneje znižuje v pričakovanju nove količine. To se zgodi ob $\lambda \in (0, 2)$ [4]. Če bi strežnik dolgo časa ostal brez zahtev, bi lahko izračunana frekvenca ($f(j-1)$) končala na zelo nizki vrednosti, s katere bi jo težko hitro dvignili. Da se to prepreči, jo omejimo med $f_{min}$ in $f_{max}$ v dodatnem pod-koraku [4].

### Nadzornik strežnika

Predlog algoritma za nadzornika strežnika s slike 3.2.

Na vsakem kontrolnem intervalu (indeksiran z '$k$') se zgodi sledeče [4].

1. Poizvemo povprečno porabo električne energije strežnika za prejšnji interval '$srvPow(k-1)$'.

2. Izračunamo, koliko energije ima strežnik na voljo (v procentih $\in [0, 100]$) od predpisane omejitve porabe. Najprej izračunamo odstopanje:

$$err(k) = \frac{srvCap - srwPow(k-1)}{maxPow - minPow} \tag{3.5.12}$$

'$err(k)$' predstavlja delež med razliko limita '$srvCap$' in porabo iz prejšnjega intervala glede na razliko med maksimalno in minimalno porabo. Nato določimo porabo električne energije '$qv(k)$' za trenutni interval v procentih:

$$qv(k) = qv(k-1) + K * err(k) \tag{3.5.13}$$

Procenti se uporabljajo, da naredimo algoritem neodvisen od konkretne porabe strežnika, ki je v vatih. '$qv(k-1)$' je delež porabe električne energije iz prejšnjega intervala. '$K$' je konstanta. Nazadnje izračunane procente še omejimo v vrednosti:

```
if (qv(k) > 100) qv(k) = 100
else if (qv(k) < 0) qv(k) = 0
```

3. Izračunani delež energije pretvorimo v ustrezen indeks strojno podprte frekvence:

```
if (qv(k) < gv_threshold)
     indFreq(k) = floor(Gain_low * qv(k) + nFreqs)
else indFreq_(k) = floor(Gain_high *(qv_threshold - 100))
```

4. Če procesor ne teče pri strojni frekvenci za trenutni interval, indeksirani z $indFreq(k)$, jo nastavi.

Algoritem je poskus linearizacije nad-linearne povezave med električno močjo (porabo) in nastavljeno frekvenco [4]. Zato se pretvorba v tretjem koraku razbije na dve linearni funkciji. Če je trenutni delež energije pod vrednostjo, ki ločuje ti dve funkciji '$qv_{threshold}$', se uporabi prva linearna funkcija z '$Gain_{low}$' koeficientom. Če je preko vrednosti, se uporabi druga linearna funkcija z '$Gain_{high}$' koeficientom [4]. Za več informacij in grafe, prosim, poglejte v vir [4].

### Nadzornik gruče

Predlog algoritma za nadzornika gruče s slik 3.2 [4]. Slika 3.3 prikazuje seštevanje potrošnje, preden jo nadzornik gruče prejme. V algoritmu potrebujemo podatke o porabi električne energije individualnega strežnika.

V vsakem '$l$'-tem kontrolnem intervalu se izvedejo sledeči koraki [4].

1. Poizvemo povprečno porabo električne energije za vsak posamezni strežnik iz prejšnjega intervala '$srvPow_m(l-1)$'. '$m$' je številka strežnika od vsega '$M$' strežnikov.

2. Poizvemo (izračunamo) porabo električne energije celotne gruče '$grpPow(l-1)$' za prejšnji interval.

3. Porazdeli proračun električne energije med strežniki sorazmerno z njihovo trenutno porabo:

$$srvCapm(l) = \frac{srvPowm(l-1)}{grpPow(l-1)} * grpCap \tag{3.5.14}$$

Algoritem upošteva energetske potrebe (zaradi sprememb v frekvenci in potrošnji) individualnega strežnika in temu ustrezno porazdeli energijo [4]. Omenili smo tudi možnost izklopa posameznega strežnika v opisu nadzornika delovnega bremena (s slike 3.3), kar je dodaten razlog za ustreznost pristopa kljub preprostosti.

## 3.5.7 Porazdelitev delovnega bremena

Naloga nadzornika delovnega bremena je minimizacija potrošnje električne energije: $min(\sum_{i=1}^{m} power_i)$. '$m$' je število strežnikov (*servers/blades* s slike 3.3). Pri tem mora upoštevati omejitve porabe na vseh nivojih in upoštevati raznolikost strojne opreme strežnikov [4].

- Lokalno mora upoštevati limit električne energije, predpisan za individualni strežnik [4]:

$$power_i \leq LOCAL\_LIMIT_i \tag{3.5.15}$$

- Za omarice s strežniki mora upoštevati limit, predpisan po posamezni omarici:

$$\sum_{i=1}^{m}(M_{iq}power_i) \leq ENCLOSURE\_LIMIT_q \tag{3.5.16}$$

'$q$' indeksira posamezno omarico in ima vrednosti od 1 do '$l$'. '$l$' je količina omaric. '$M_{iq}$' je matrika, v kateri z '0' prikažemo ločenost od omarice in z '1' pripadnost strežnika v omarico [4].

- Za celotno gručo se upošteva trenutni limit za gručo [4]:

$$\sum_{i=1}^{m} power_i \leq GROUP\_LIMIT \tag{3.5.17}$$

Glavni del algoritma je narediti razpored *VM* med strežniki:

$$\sum_{j=1}^{n} X_{ij} = N \tag{3.5.18}$$

'$X_{ij}$' je matrika pripadnosti *VM* k strežniku, kar simbolizira '1'. '$N$' je količina virtualnih naprav [4]. Pomaga, če si sistem predstavljate kot hibrid arhitektur s slike 3.3, ki upravlja s potrošnjo energije na strojni opremi, in arhitekture s slike 3.4, ki upravlja z aplikacijami. Obe arhitekturi se združita v nadzorniku učinkovitosti.

Ko je razpored izbran, mora ustrezati pogojem zanj. Breme, ki ga predstavlja migracija *VM*, ne sme biti preveliko:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{migration}|X_{ij} - X_{ij}^0| \leq OVERHEAD_{migration} \tag{3.5.19}$$

'$\alpha_{migration}$' predstavlja konstantno breme, ki ga migracija povzroča. '$X_{ij}^0$' je razpored, ki je bil uporabljen v prejšnjem kontrolnem intervalu. Vsili omejitev števila migracij *VM* v vsakem kontrolnem intervalu [4].

Ker želimo migracijo *VM*, moramo ugotoviti vpliv migracije na strežnike. Zato ocenimo obremenjenost procesorja za vsak strežnik '$r_i$':

$$r_i = min(1.0, \sum_{j=1}^{n} X_{ij}U_j(1 + \alpha_{virtual}))_i. \tag{3.5.20}$$

Obremenitev procesorjev posameznih strežnikov je omejena na 100 %. '$U_j$' je matrika s povprečnimi obremenitvami procesorjev znotraj strežnika pod nastavljeno

frekvenco. '$\alpha_{virtual}$' predstavlja dodatno breme zaradi *VM* ob bremenu aplikacije [4].

Ko so obremenitve procesorjev izračunane, jih preverimo, da ne presegajo referenčne obremenitve procesorja '$\bar{r}_i$', ki jih postavlja lokalni nadzornik porabe električne energije (LNPEE s slike 3.3): $r_i \leq \bar{r}_i$ [4]. Obremenitve procesorja morajo biti "umerjene" na dejansko obremenjenost pri maksimalni mogoči frekvenci. Dva strežnika z istima relativnima obremenitvama procesorja, vendar z različnima nastavljenima frekvencama procesorjev nista primerljiva [4]. In ob različni strojni opremi je "umerjenost" na dejansko obremenitev procesorja nujna.

Nazadnje predvidimo porabo električne energije iz ocenjenih obremenitev procesorjev na strežnikih: $power_i = rise_o * r_i + base_0$. '$base_0$' predstavlja začetno potrošnjo, ko strežnik ne izvaja aplikacij in je obremenitev procesorja približno nič. '$rise_0$' predstavlja linearni faktor naraščanja potrošnje z večanjem obremenjenosti procesorja. Oba parametra sta izmerjena pod maksimalno nastavljivo frekvenco zaradi poenostavitve problema sporočenih obremenitev procesorja [4].

Izračunano porabo električne energije preverimo glede na limit porabe, naveden na začetku opisa optimizacije. Če na strežniku ne teče nobena virtualna naprava, se sklepa, da bo kmalu izklopljen in njegova potrošnja je nič. V praksi nekaj strežnikov brezdelno čaka v stanju pripravljenosti za primer nenadnega prihoda velike količine zahtev [4].

### Iskanje razporeda virtualnih naprav

Doslej smo našteli enačbe in omejitve pri optimizacijskem problemu. Ob velikem številu strežnikov problem lahko postane praktično neizvedljiv (*infeasible*) v času kontrolnega intervala [4]. Posledica je, da je kontrolni interval za nadzornika delovnega bremena daljši od ostalih nadzornikov. Tudi selitev *VM* in izklop in vklop strežnika lahko zahtevata daljši čas, odvisno od izvedbe. Za selitev so v viru [4] le vklopili/izklopili že vnaprej pripravljeno kopijo *VM* z ustreznim nivojem aplikacije za nameščenim.

Za reševanje (preiskovanje možnih razporeditve *VM*) se sicer uporabljajo hevristične metode, kot so na primer genetski algoritmi [4]. Ne bodo poiskali najboljše rešitve, vendar se s pravilnim nastavitvami časa iskanja in kombiniranja nedokončnih rezultatov lahko približamo najboljši rešitvi.

### 3.5.8 Testi koordinacije

Testi so v viru [4] izvedeni na simulaciji s pomočjo dnevniških zapiskov (*trace*) prihodov zahtev, ki so jih tudi združili, da bi umetno ustvarili situacijo z večjo količino zahtev. Koordiniran sistem je dosegel 64 % prihranka energije v primerjavi s sistemom brez kontrole porabe električne energije (*baseline*). Zmogljivost sistema je padla za 3 % ob 5 % kršitvi proračuna električne energije. V primerjavi s sistemom brez koordinacije nadzornikov, kjer je zmogljivost padla 12 % in proračun energije kršen 7 % časa izvajanja [4].

Pri vseh količinah zahtev se je bolje odrezal sistem s kontrolo porabe električne energije (koordiniran ali ne) proti sistemu brez kontrole porabe z večjimi relativnimi prihranki ob nizkem bremenu [4]. Če primerjamo koordinacijo proti ne-koordinaciji, so relativni prihranki večji ob visokih bremenih [4].

Pri tem je bolj pomemben razpon nastavljivih strojnih frekvenc. Te so bolj pomembne, kot dolžine kontrolnih intervalov uporabljenih v nadzornikih [4]. Pri intervalih bi si sicer mislili, da krajši, kot so, bolj stroga je kontrola in krajše so prekoračitve električnega proračuna. To je pomemben parameter, toda večji razpon nastavljivih frekvenc prinaša večjo priložnost za varčevanje. Kot primer ima moj procesor iz razdelka 3.2 od 1,4 do 4 GHz in štiri-jedrni Xeon iz testov v razdelku 3.4 le 1,6 in 1,86 GHz. Dolžine kontrolnih intervalov, ki so jih uporabili v simulaciji v viru [4] so:

- 1 za nadzornika učinkovitosti (NU),
- 5 za lokalnega nadzornika (LNPEE),
- 25 za nadzornika omarice (NPEEO),
- 50 za nadzornika gruče (NPEEG) in
- 500 za nadzornika delovnega bremena (NDB).

Vse nadzornike najdemo na sliki 3.3. Pri tem sklepam, da so enote v sekundah, vendar enote v viru [4] niso navedene.

## 3.6 Zanesljivost diskov in njihova poraba energije

Zanesljivo hranjenje podatkov predstavlja vse bolj pomembno področje za velike računalnike sisteme. Potrebe po vedno večjem prostoru za shranjevanje podatkov zahtevajo vedno večje število trdih diskov [6]. S tem naraste tudi verjetnost, da eden izmed njih odpove. Hkrati se povečuje tudi kapaciteta diskov, gostota podatkov na njih. Vendar izgube podatkov niso sprejemljive [6]. Bi sprejeli izginjanje komentarjev, slik in videov na spletnih straneh Facebook, Twitter, Youtube? Ali izgubo rezultatov raziskav, ki so vzele mesece na superračunalniku?

Neposredne kopije podatkov ali dodatni (meta)podatki, ki nam omogočijo rekonstrukcijo izgubljenih ali okvarjenih, predstavljajo nujno redundanco poleg bistvenih podatkov [6]. Oboje morajo diski hraniti. Potrebujemo torej še več prostora za hranjenje podatkov, še več trdih diskov, da povečamo zanesljivost sistema hranjenja podatkov. Diski porabijo do 27 % energije podatkovnega centra [6]. V prihodnosti lahko pričakujemo, da bo ta delež še večji.

To nas sili k vpeljavi tehnik varčevanja z energijo na diskih. Trenutno najbolj preprosta in v splošni uporabi je izklop diskov po preteku od uporabnika nastavljenega časa neaktivnosti (*timed-off shutdown*). *RAID* se uporablja za reševanje odpovedi enega (*RAID*-5) do dveh (*RAID*-6) trdih diskov [6]. Za odkrivanje latentnih okvar sektorjev (*latent sector errors - LSE*) se uporablja pregled diska ali preverjanje po vpisu [6].

V vseh računalniških sistemih želimo izboljšati zanesljivost in hkrati varčevati z električno energijo. Želimo združiti tehnike povečevanja zanesljivosti diskov in varčevanja z energijo [6]. Za nemoteno delovanje v aktivnem času diskov se obe vrsti tehnik, ne glede na to, katera je uporabljena v praksi, izvajata v času neaktivnosti diskov. Vpraševanje je, kolikšen delež časa neaktivnosti nameniti tehniki za večanje zanesljivosti in preostanek tehniki za varčevanje z energijo [6]?

Pri večanju zanesljivosti aktivno beremo z diska, torej podaljšujejo čas aktivnosti diska, da odkrijemo *LSE* [6]. Področje energetske učinkovitosti se ukvarja ravno z obratnimi tehnikami, kjer želimo podaljšati čas neaktivnosti, diske upočasniti ali začasno izklopiti. Medtem ko so tehnike samostojno poznane in dobro raziskane, interakcijo med tehnikami za zanesljivost in za varčevanje še vedno raziskujejo [6].

### 3.6.1 Večanja zanesljivosti diskov

Cilj teh tehnik je podaljšanje povprečnega časa do izgube podatkov (*mean time to data loss - MTTDL*), ki je mera za zanesljivost [6].

#### Latentne okvare sektorjev

Skrite, še neodkrite, latentne okvare sektorjev poznamo tudi pod imenom neobnovljive okvare ob branju (*unrecoverable read errors*). Nastanejo po uspešnem vpisu podatkov v sektor in so odkrite ob branju ali prej s tehniko za večanje zanesljivosti.

Vzroki za okvaro so neposredno fizični (opraskana površina, izrabljenost premičnih delov). Posredno lahko izvirajo iz drugih vzrokov, na primer okvara krmilnega vezja, vdor prahu na površino diska, defekti med izdelavo, vibracije med prenosom diska, stik bralno-pisalne glave s ploščo. Pri tem lahko en vzrok vodi v drugega in večja količina *LSE* je lahko pokazatelj okvarjene površine diska. V najslabšem primeru izgubimo vse podatke.

Pogostost okvar diskov in njihovih zamenjav se iz praktičnih podatkov razlikuje od proizvajalčevih navedenih povprečnih časov do odpovedi diska (*mean time to failure - MTTF*). Pri tem ni velikih razlik med tipi diskov [6]. Pomembni dejavniki so starost diska in pogoji obratovanja. Starejši diski imajo večjo verjetnost za pojav *LSE* [6], prav tako diski, ki obratujejo v prašni okolici ali so močneje obremenjeni. Ugotovili so prostorsko in časovno zgoščenost več *LSE*. Torej so bile povzročene iz enega dogodka kot je praska ob dotiku bralno-pisalne glave ob ploščo [6].

Ti podatki kažejo, da je modeliranje pojavov *LSE* po Poissonovi porazdelitvi napačno. Ustrezno nastavljena Paretova porazdelitev se bolje ujema s praktičnimi podatki odpovedi diskov [6]. Uporaba različnih porazdelitev povzroča probleme pri primerjavi rezultatov posameznih raziskav.

#### Polje poceni diskov z redundantnimi podatki

*Redundant Array of Inexpensive Disks - RAID* so polja diskov, kjer z različnimi porazdelitvami blokov podatkov in blokov metapodatkov med diski poskrbimo za redundanco. *RAID*-0 nima redundance. Pri *RAID*-1 so metapodatki kopije blokov. Pri *RAID*-2 se med diski porazdeljujejo biti in odstranitvene kode (oblika

*error correction & check - ECC*). Pri *RAID*-3 se porazdeljujejo bajti s paritetnimi biti na dodeljenem disku. Pri *RAID*-4 se porazdeljujejo bloki podatkov s paritetnim blokom na dodeljenem disku. Vsi od 2 do 4 niso v široki uporabi zaradi pomanjkljivosti [66, 15, 16]. Zaradi redundance, razen pri *RAID*-0, je potrebno več diskov, kar pomeni večjo porabo energije [6].

Na večjih računalnikih sistemih se uporablja *RAID*-5. Zahteva najmanj 3 diske, kjer se bloki podatkov in blok s paritetnimi biti krožno permutirajo med diski. Paritetni blok sedaj ni več na istem disku, kar poveča zanesljivost. Če eden odpove, ga nadomestimo. Pri tem se podatki obnovijo iz podatkov in paritetnih bitov na preostalih diskih.

Toda med obnovo lahko naletimo na *LSE*. V najboljšem primeru bo odsek, ki ga ne more obnoviti, preskočen. Vseeno smo izgubili podatke. V najslabšem primeru bo celoten proces neuspešen in bomo izgubili podatke celotnega polja, razen če imamo nekje kopijo vseh podatkov [55]. Ta scenarij je tudi glavni zagovor uporabe *RAID*-6, ki *RAID*-5 nadgradi s dodatnim blokom paritetnih bitov na odsek.

### Prepleteno preverjanje paritete

*Interleaved Parity Check - IPC* je mehanizem z redundanco znotraj diska, ki rešuje problem pojavitve *LSE* [6]. Pred vpisom na disk se segment sledečih si sektorjev logično porazdeli v matriko in doda na dno matrike sektor s paritetnimi biti za posamezno kolumno. Med splošne podatke v *RAID*-5 se vpiše segment, ki je že "prepletenica" podatkov in metapodatkov, poleg tega segmenta *RAID*-5 doda še svoj paritetni blok [6]. Glede na analizo in simulacije ima uporaba *IPC* minimalni vpliv na obremenjenost diskov; tako *RAID*-5 z *IPC* postane primerljiv *RAID*-6 [6].

### Pregled

*Scrubbing* je periodično branje celotnega diska z namenom odkritja *LSE*, preden bomo potrebovali vsebino pregledovanega sektorja [6]. Zato se ga uporablja na *RAID*. Če odkrijemo *LSE*, lahko popravimo vsebino preko *RAID* in uporabimo ostale podatke in paritetni blok [6]. Sektor bo obnovljen v rezervnem prostoru diska, ki je namenjen nadomeščanju okvarjenih sektorjev. Pregled praviloma opravimo vsak dan ali najkasneje na vsaka 2 tedna [6].

*Staggered scrubbing* za razliko od navadne oblike segmente v disku zbere v regije. *Scrubbing* pregleduje najprej znotraj regij [6]. Ker so *LSE* lokalizirane, pregled regij izboljša možnost, da enega izmed okvarjenih sektorjev hitro odkrijemo [6]. Za merjenje uspešnosti so predlagali povprečni čas od okvare sektorja (pojava *LSE*) do odprave (*Mean Latent Error Time - MLET*). *MLET* je do 30 % krajši od normalne zaporedne verzije in povprečen čas iskanja *LSE* do 40 % krajši [6]. Obstaja torej možnost za optimiziranje vrstnega reda pregleda (*scrubbing*) segmentov diska.

*Accelerated scrubbing* izkorišča informacijo, da se od 20 % do 50 % okvar sektorjev nahaja v prvih 10 % logičnega prostora diska [6]. Odkrivanje *LSE* se pri tej vrsti začne v omenjenem območju. Glede na razdaljo med okvarami po času vsi diski doživijo okvaro sektorja na enako dvotedensko obdobje, če vse *LSE* razporedimo čez življenjsko dobo diska [6].

### Redundanca znotraj diska

*Intra-disk redundancy* je namenjen obnovi ob pojavu *LSE* iz kopije podatkov, ki so shranjene znotraj istega diska [6]. Nastaviti je potrebno ustrezno število kopij, da ni velikega vpliva na performanse diska. Predstavlja cenejšo alternativo, če kopije ustvarjamo znotraj *RAID*-5 v primerjavi z *RAID*-6, ki bi zahteval dodaten disk [6]. Ta pristop odpravi potrebo po rednem pregledovanju diska (*scrubbing*) [6], vendar je pregled še vedno potrebno občasno opraviti, ker je lahko veliko število *LSE* pokazatelj težav z diskom.

### Zakasnjeno preverjanje po vpisu na disk

*Idle Read After Write - IRAW* je oblika preverjanja vpisanih podatkov. Po vpisu ohranja kopije vpisanih sektorjev v medpomnilniku diska do časa neaktivnosti diska. Tedaj primerja vsebino vpisanih sektorjev in medpomnilnika [6]. Takojšnje preverjanje vpisanih podatkov slabo vpliva na performanse diska. Ker se večina *LSE* pripeti kmalu po vpisu, se s tem procesom izognemo večini *LSE* [6]. Izboljšana zanesljivost je primerljiva s pregledovanjem diska (*scrubbing*), vendar se lahko disk začne odzivati počasneje, zaradi zasedenosti medpomnilnika diska [6].

### 3.6.2 Varčna uporaba diskov

Za varčevanje z energijo se uporablja namerni izklop celotnega računalnika, izklop zaslona, izklop diska ali preklop v stanje nizke porabe (*sleep*) po preteku nastavljenega časa neaktivnosti uporabnika [6]. Izziv je prepoznati čase neaktivnosti, ki so zadosti dolgi, da se prihranjena energija (vsaj) izenači (*break-even time*) z energijo, uporabljeno za zagon (*spin-up*) diska, ki lahko traja nekaj sekund [6]. Agresivni prediktor, ki bi to storil hitreje kot tehnike, temelječe na nastavljenem pretečenem času, je v raziskavi. Kontrolo nad diski bi bilo najbolje prepustiti programerjem, saj najbolje poznajo aplikacijo in kaj uporabnik potencialno dela [6].

#### Masivno polje neaktivnih diskov

*Massive Array of Idle Disks - MAID* je tip polja, kjer se podatki skopirajo v diske, ki imajo vlogo medpomnilnika (*cache*) in so aktivni ves čas. Preostali diski (*non-cache*) so lahko zato dalj časa neaktivni [6].

#### Zgoščanje popularnih podatkov

*Popular Data Concentration - PDC* je še boljši pristop, kjer pogosto dostopane podatke premaknemo na izbrano podskupino diskov. V primerjavi z *MAID* ne zahteva dodatnih (*cache*) diskov in prihrani več energije [6]. Vendar *PDC* spreminja zanesljivost sistema diskov, ker podskupino diskov s popularnimi podatki dodatno obremeni, kar zahteva dodatne raziskave [6].

#### Potrošnje energije v polju diskov

Da performanse polja diskov sledijo performansam množice procesorjev se uporablja veliko več diskov, kot je procesorjev, v razmerjih na primer 35 ali 21 diskov na en procesor. V takih razmerjih diski potrošijo več kot štirikratnik energije procesorja [6].

S pravilno izbiro datotečnega sistema in njegovimi nastavitvami lahko prihranimo od 5,4 % do 840 % energije več od privzetih nastavitev [6]. Strežniki imajo predvidljive vzorce obremenjenosti z zahtevami, kar pomaga izbrati nastavitve datotečnega sistema. Vendar univerzalnega datotečnega sistema, ki bi dobro deloval v vseh situacijah, ni [6].

### Varčevanje z migracijo popularnih blokov podatkov

V primeru več-hitrostnih diskov se lahko hitrost prilagodi, da ustreza obremenitvi, količini zahtev [6]. Dodatni mehanizem je uporaba ustrezne zamenjevalne strategije, da popularni bloki podatkov ostanejo čim dalj časa v pomnilniku in diski lahko ostajajo dalj časa v stanjih nizke porabe (neaktivnosti) [6]. Aktualna raziskovana zamenjevalna tehnika je bila z dinamično prilagodljivimi pomnilniškimi particijami, ki hranijo podatke iz različnih diskov [6].

Glavni problem pri *PDC* je, da zahteva čas za zbiranje statistike dostopov do datotek, da se zabeleži statistika dostopov in prepozna, kateri podatki so popularni [6]. Pristop ne deluje dobro za strežnike, kjer se ne dostopa do celotnih datotek. Pri strežnikih s podatkovno bazo, kjer se dostopa do blokov podatkov znotraj datotek, zato uporabljajo *program counter - PC*, da identificirajo popularne bloke podatkov [6].

Drug pristop je z uporabo virtualizacije. Iz vsakega logičnega diska (*volume*) se izbere nekaj blokov podatkov in združi v "delovno zbirko" [6]. Ta zbirka je skopirana na podskupino diskov. Vsi dostopi se z virtualizacijo preusmerijo na podskupino diskov (ali enega) [6]. Ostali diski so upočasnjeni ali v stanju nizke porabe. Pri tem celotni podatki (datoteke) niso premaknjeni med diski za razliko od *PDC* [6].

### Varčevanje z reprodukcijo na oddaljenih lokacijah

Pri oddaljenih varnostnih kopijah je smiselno zakasniti izvedbo sprememb na varnostni kopiji [6]. Pošljemo le spremembe, ki morajo biti narejene na že obstoječi kopiji. Uporabimo stiskanje sprememb (*log folding*) nad dnevnikom sprememb, kjer se več sprememb nad istim podatkom strne v eno samo spremembo [6]. V viru [6] so zapisi v dnevniku sprememb vsebovali fizične naslove do diskov, zato so lahko sortirali spremembe v tako imenovana "okna", da je naenkrat čim več dostopov do istega diska, ki uvaja spremembe na svojem delu kopije. Z zakasnitvijo, stiskanjem in sortiranjem so uspeli doseči 80 % do 85 % prihranka energije [6].

#### Nove tehnike shranjevanja

Poskusno so vgradili procesor z nizko porabo električne energije neposredno na *solid state disk - SSD* [6]. Na standardnih sistemih potrošnja energije na procesorju raste nad-linearno z njegovo hitrostjo, kar naredi uravnavanje porabe na takih sistemih neučinkovito [6]. Z vgradnjo procesorja so dobili najboljšo možno kombinacijo hitrosti in potrošnje energije.

Vendar so avtorji raziskave priznali, da so naslavljanje preko parov ključa in vrednosti prilagodili posebej za lastnosti pomnilnika *Flash* [6]. Poskus je vsaj pokazal poceni alternativo tradicionalnim trdim diskom. Vendar imajo *SSD* probleme z manjšo kapaciteto, omejenim številom vpisov v pomnilnik *Flash*, posebno ob intenzivnem vpisovanju na isto lokacijo, kar lahko poslabša odzivnost diska bolj kot na mehanskih diskih z vrtljivimi ploščami. Zato *SSD* niso v širši uporabi [6].

Drugi poskus je tehnika, imenovana *Hibernator*. Preko algoritma ugotovi popularnost podatkov in obremenitev diskov [6]. Nastavlja hitrost diskov in preklopi v stanje nizke porabe. Podatke s podobnim vzorcem dostopov premakne na disk z ustrezno hitrostjo [6]. Najpogosteje dostopani so premaknjeni na disk, ki se vrti s polno hitrostjo. Manj pogosto dostopani so na počasnejših diskih, ostali diski so v stanju nizke porabe (*standby*). Da se aplikacija izvede pravočasno, lahko disk tudi pospeši [6].

### 3.6.3 Kombinacija zanesljivosti in varčevanja z energijo

Če želimo kombinirati tehniki varčevanja z energijo in izboljšanja zanesljivosti diska, potrebujemo merljivo vrednost za učinek obeh tehnik v kombinaciji. Vir [6] je predlagal uporabo zmnožka privarčevane energije in povečanja zanesljivosti (*Energy-Reliability product - ERP*). Alternativa je uporaba količine dodatne energije, potrebne za izboljšavo zanesljivosti, zmnožene s povečanjem zanesljivosti, kar so uporabili v viru [6].

V viru [6] so uporabili izklop po preteku nastavljenega časa za varčevanje z energijo in *scrubbing* za izboljšanje zanesljivosti diskov. Izboljšava zanesljivosti pri pregledu (*scrubbing*) površine diska je podaljšanje časa do izgube podatkov $\Delta MTTDL$. '$T_{scrub\_period}$' predstavlja čas, ki preteče med dvema začetkoma pre-

gleda površine diska.

$$ERP = \Delta Energy * \Delta MTTDL \propto \Delta Energy * \frac{1}{T_{scrub\_period}} \tag{3.6.1}$$

V viru [6] so uporabili posnetke (*trace*) praktične uporabe aplikacij, da so simulirali dostope do diskov. *ERP* je uspel meriti interakcijo med tehnikama. Tako so s pomočjo *ERP* identificirali delež časa neaktivnosti (*idle period*), ki mora biti namenjen pregledu površine diska, da dobimo najboljše iz kombinacije. Pri tem moramo paziti, katero tehniko uporabimo v izbranem času neaktivnosti [6]. Ker so bili nekateri časi neaktivnosti prekratki (krajši od *break-even* časa) in izklop diska ni bil smiseln, so te čase namenili za pregled površine diska [6].

V viru [6] so testirali sledeče porazdelitve časov neaktivnosti med tehnikama:

1. V začetku aplikacije so vse čase neaktivnosti namenili za pregled površine, nato so vedno izklapljali disk po preteku časa. Rezultat je slaba zanesljivost in tudi *ERP*.

2. Kratke čase neaktivnosti so namenili pregledu površine. Dolgi časi neaktivnosti so bili uporabljeni za izklop diska po preteku časa. Rezultat je bil, da se je pregled površine izvedel premalokrat in *ERP* je imel vrednost blizu nič.

3. Nazadnje so poskusili izmenično menjavanje med tehnikama. Prvi čas neaktivnosti bi bil torej namenjen pregledu površine, drugi izklopu diska po preteku časa, tretji spet pregledu površine in tako naprej. Pri tem so prekratke čase prepustili pregledu površine. Ta pristop dodeljevanja tehnik v čase neaktivnosti je dal dobre rezultate v *ERP*.

*ERP* je zgolj začetek raziskav odnosov med zanesljivostjo in varčevanjem z energijo in celo vir [6] priznava potrebo po nadaljnjih raziskavah, na primer raziskavah drugih kombinacij med tehnikama zanesljivosti in varčevanja, raziskavah delovanja *ERP* na več-hitrostnih diskih, *SSD* ali hibridnih diskih in tako dalje.

## 3.7 Varčevanje s pomočjo prevajalnikov

Strojna oprema ponuja mehanizme za znižanje porabe električne energije, kot so *DVFS*, stanja aktivnosti ali izklop nepotrebnih resursov (to so diski v tem raz-

delku), pod pogojem, da mehanizme sistemski programi tudi uporabijo [3]. Prevajalnik lahko analizira delovanje programa (*program behavior*), vzorce dostopa (*data access patterns*) do pomnilnika in zahtevane resurse (*resource requirements*) [3].

Na porabo električne energije lahko vpliva neposredno z izklopom nepotrebnih virov ali posredno s podaljšanjem časa med uporabo virov [3]. V viru [3] opisujejo *framework* za prevajanje (*compilation framework*), ki spremeni kodo programov in vstavi upravljanje z resursi, da zniža porabo.

### 3.7.1 Statično prevajanje

Prevajalnik določa hitrosti diskov, razpored podatkov in razdaljo pred-prevzema (*pre-fetch distance*) za podatke z diska ob prevajanju aplikacije. V viru [3] predstavijo predlog za statični *framework* prevajalnika, ki to zmore.

#### Razdalja pred-prevzema

Znanstvena aplikacija, ki se izvaja na rezervirani množici procesorjev [7], ima v sebi več vgnezdenih zank [3]. Program brez prilagoditve kode izmenično prehaja med časom dostopa do diska in časom računanja s podatki. Ti časovni intervali so zaporedni. Izgubimo '$T_d$' ciklov za prenos bloka z diska [3].

*Framework* doda operacijo pred-prevzema (*prefetch*). V času računanja s podatki za trenutni interval sočasno nalaga podatke z diska, preden bodo ti potrebni pri računanju v naslednjem intervalu [3]. Vzporednost obdelave podatkov in nalaganja z diska sicer zmanjša čas izvajanja aplikacije, s čimer prihrani skupno porabo energije, toda ne zniža porabe energije na diskih [3].

Če želimo pred-naložiti blok iz diska, preden ga bomo potrebovali, moramo poznati razdaljo pred-prevzema '$d$', ki je število iteracij zanke, potrebnih, da skrijemo vhodno-izhodno latenco [3].

$$d = \left\lceil \frac{T_d}{s + T_{pf}} \right\rceil \tag{3.7.1}$$

V enačbi 3.7.1 je '$T_d$' ocenjena vhodno-izhodna latenca (v ciklih) in je odvisna od hitrosti vrtenja diska [3]. '$T_{pf}$' je režija (*overhead*) potrebna za operacijo

pred-prevzema (v ciklih). '$s$' je število ciklov po najkrajši poti skozi telo zanke [3].

Da znižamo porabo električne energije na diskih, lahko znižamo hitrost vrtenja diskov. Vir [3] v svojih virih navaja, da je poraba električne energije diska kvadratno proporcionalna hitrosti vrtenja diskov. Zato so z začetnih 12000 vrtljajev na minuto (*revolutions per minute - RPM*) nastavili diske na 6000 *RPM* [3]. Razdalja pred-prevzema '$T_d$' se je podvojila, vendar omogočila obratovanje diskov pri znižani porabi električne energije. V času, ko disk ne izvaja pred-prevzema, je v stanju neaktivnosti (*idle*) [3].

### Primer preoblikovanje kode

Prevajalnik mora preoblikovati kodo, da lahko izkoristimo pred-prevzemanje. Polje '$V_1$' se nahaja na disku (*disk-resident array*). Primer kode [3]:

```
for i=0 to N-1 {
    for j=0 to M-1 {
        ... V1[i][j] ...
    }
}
```

Ta primer naj bi po uporabi predloga za *framework* po vključitvi operacij pred-prevzema imel kodo, podobno [3]:

```
for i=0 to N-1 {
    j = 0;
    // zacetek, pred-prevzamemo blok
    PF( &V1[i][j] );
    // stabilno stanje, pred-prevzem bloka in racunanje
    for jj=0 to M-1-d, za korak b1 {
        PF( &V1[i][j+d] );
        for j=jj to jj+b1 {
            ... V1[i][j] ...
        }
    }
    // zakljucek, racunanje z zadnjim blokom
```

```
    for j=M-d to M-1 {
        ... V1[i][j] ...
    }
}
```

### Hitrosti diskov

Aplikacije si predstavljamo kot množico vgnezdenih zank P = ($LN_1, LN_2 \ldots LN_s$). Program se uporabi kot vhod v algoritem za izbiro hitrosti diskov, vključno z hitrostmi diskov, ki so na izbiro RPM_nivoji = (min ... max). Vrne izbrane hitrosti za vsako polje, shranjeno na diskih [3].

Za začetek izpraznimo množico predlaganih hitrosti '$G$' za posamezno polje '$V_k$', ki je shranjeno na enem izmed diskov. Določi se režija pred-prevzema '$T_{pf}$' (sklepam, da podatek posreduje prevajalnik v predlagani *framework*, ker to v viru [3] ni navedeno).

```
T_pf = rezija pred-prevzema
for V_k ∈ ν {
    G[V_k] = 0;
}
```

Za vsako izmed gnezd zank '$LN_i$' se izmeri čas pridobitve bloka z diska '$T_d$' pod možnimi hitrostmi diskov. Nazadnje dobimo predlagane hitrosti diskov za vsako referenco na polje [3]. Ker gremo od minimalne hitrosti RPM k maksimalni, postane razdalja pred-nalaganja vse krajša in pogoj ni izpolnjen za majhna števila iteracij skozi gnezdo zank. Ob premajhnem številu iteracij pred-prevzemanje ni mogoče [3]. Število ciklov, potrebnih za izvedbo zanke '$s_i$', v *framework* posreduje prevajalnik (ponovno sklepam, ker v viru [3] to ni navedeno).

```
for vsako gnezdo zank LN_i ∈ P {
    s_i = stevilo ciklov skozi gnezdo zank LN_i
    for vsako hitrost RPM ∈ RPM_nivoji {
        // sklepaj, da je a_i element iz polja V_k
        for vsako referenco na polje a_i {
            izracunaj latenco diska T_d(RPM);
            izracunaj razdaljo pred-prevzema
```

$d_{RPM} = \left\lceil \frac{T_{d(RPM)}}{s+T_{pf}} \right\rceil$;
if ($d_{RPM}$ > skupno st. iteracij skozi $LN_i$)
G[$V_k$] = G[$V_k$] $\cup$ RPM;
}
}
}

Ker lahko dostopamo do istega polja iz različnih gnezd zank, se izbere največja hitrost diska za to polje [3]. Preučevano polje dodamo k skupini z ustrezno hitrostjo.

for vsako polje $V_k \in \nu$ {
$izbrani_{RPM}$ = { RPM | RPM $\in$ G[$V_k$] in je MAX(G[$V_k$]) };
$skupina_{(izbrani_{RPM})}$ = $skupina_{(izbrani_{RPM})}$ $\cup$ $V_k$;
}

### Razpored podatkov

Zadnji korak v predlogu je dodeljevanje polj na diske z ustrezno nastavljeno hitrostjo. Vhod v algoritem za dodeljevanje polj k diskom vsebuje kodo programa P = ($LN_1, LN_2 \ldots LN_s$) in v prejšnjem koraku grupirana polja glede na potrebno hitrost diskov '$skupina_{(izbrani_{RPM})}$'. Rezultat tega bo razpored podatkov (*data layout*) [3].

Za začetek se določi število diskov '*vsi_diski*', številka diska, od katerega začnemo *data striping* '*init_disk*', število dostopov do polja $V_k$ '*teza*[$V_k$]' in celotno število dostopov v programu '$vsa_teza$'. Izvor teh informacij v viru [3] ni pojasnjen, sklepam, da jih posreduje prevajalnik v predlagani *framework*.

vsi_diski = stevilo diskov, ki kolikor jih je na voljo;
init_disk = 0;
teza[$V_k$] = stevilo dostopov do polja $V_k$;
vsa_teza = stevilo vseh dostopov v programu;

Izmed diskov z ustrezno hitrostjo, polja podatkov, ki imajo več dostopov, dobijo večjo težo in z njo proporcionalno tudi več diskov z enako hitrostjo, med katerimi so polja porazdeljena (*data striping*) [3].

```
for vse hitrosti RPM ∈ RPM_nivoji {
    for vsako polje V_k ∈ skupina_(RPM) {
        sum += teza[V_k];
        cez_n_diskov(V_k) = vsi_diski * ⌈sum / vsa_teza⌉;
        vsi_diski -= cez_n_diskov(V_k);
    }
}
```

Sedaj porazdelimo polja med diske na podlagi podatka, čez koliko diskov se bo polje porazdelilo '*cez_n_diskov*' (*stripe factor*) [3].

```
for vsako polje V_k ∈ skupina_(RPM) {
    zacetni_disk(V_k) = init_disk;
    init_disk += cez_n_diskov(V_k);
}
```

S tem smo določili za posamezno polje potrebne podatke za porazdeljevanje (*data striping*) [3]:

- Na katerem disku se polje začne ('*zacetni_disk*'),
- čez koliko diskov se porazdeljuje ('*cez_n_diskov*') in
- velikost delca polja.

Velikost delca polja lahko izračunamo glede na velikost polja: $\frac{velikost\_polja}{cez\_n\_diskov}$. Sedaj bi morali preoblikovati kodo, kot smo podali v primeru. Razdalja za pred-prevzem je bila izračunana v prejšnjem pod-naslovljenem koraku [3].

### 3.7.2 Dinamično prevajanje

Drugi vir v viru [3] predstavi *framework*, ki upravlja z diski dinamično v času izvajanja. Primer dinamičnega prevajalnika je prevajanje in nato tolmačenje (interpretacija) v virtualni napravi za programski jezik Java [38]. *Framework* sestavljajo [3]:

- dinamično optimizirajoč prevajalnik/povezovalnik,
- visoko-nivojna vhodno-izhodna knjižica (*high-level I/O library - HLIOL*),

- sistem mini-podatkovne baze ali meta-podatkovni upravitelj,
- upravitelj razporeditve (podatkov).

*Framework* naredi dostop do množice podatkov abstrakten in poenoten, neodvisen od lokacije in tipa medija [3]. Namenjen je vzporednim arhitekturam shrambe, ki med nivoji diskov vsebujejo podatkovne strežnike (*network-attached storage - NAS*), omrežne diske (*storage area network - SAN*), aktivne diske ali tudi terciarno shrambo (tračni sistem) [3].

*HLIOL* vsebuje optimizacijske rutine za dostop do vzporedne arhitekture shrambe. Te koordinirajo procesorje in diske z upravljanjem pretoka množic podatkov med njimi [3]. *HLIOL* vsebuje tudi rutine za prenos podatkov med komponentami shrambe (spremenijo razpored podatkov v shrambi). *HLIOL* je namenjena več-procesorskim sistemom in zahteva identifikacijsko številko procesorja v svojih rutinah [3]. Poizvedbene rutine, poslane k meta-podatkovnemu upravitelju, za trenutnimi lokacijami množice podatkov in ohranitev istega imena za množico podatkov omogočijo abstrakcijo in poenostavijo prenos množice podatkov po arhitekturi shrambe [3].

Meta-podatkovni upravitelj hrani [3]:

- Vzorce dostopov do množic podatkov in odnose med njimi, na primer, katere množice se naslavljajo skupaj.
- Statistiko performans vhodno-izhodnih dostopov.
- Trenutne lokacije množice podatkov ali delcev od množice podatkov (*subfiling*, ki je podoben *data striping*).
- Cene (čase), potrebne za izvedbo rutin iz *HLIOL*.

Upravitelj razporeditve ureja razpored podatkov v hierarhiji shrambe [3]. Izvaja prenose podatkov med enotami shrambe ali spremembe na množici podatkov, ki je že porazdeljena (*data striped*) med več komponentami shrambe. Za primer je sprememba velikosti delca (*strip size*) na posameznih enotah shrambe [3]. Upravitelj razporeditve lahko dobi več zahtev po spremembi razporeditve od prevajalnika iz več aplikacij. Upravitelj lahko zavrne zahtevo po prerazporeditvi, če optimizacija ni koristna globalno [3].

### Opis prevajalnika s primeri delovanja

Prevajalnik vsebuje 4 komponente: dinamični prevajalnik, dinamični povezovalnik, spremljevalca performans (*performance tracer*), krmilno enoto (*steering unit*) [3].

Spremljevalec performans iz meta-podatkovnega upravitelja pridobi vzorce dostopov. To so podatki, kot so: način dostopa (bralen ali večinsko bralno-pisalen), pogostost, s katerimi množicami podatkov je trenutna množica dostopana skupaj (*temporal affinity*) in podobne [3]. Prav tako pridobi statistiko performans, kot so število dostopov do diskov, na trak..., zgrešitve v pomnilniku, čas vhodno-izhodne operacije in potrošnja energije v komponentah shrambe. Ko zbere te informacije, jih posreduje krmilni enoti [3].

Krmilna enota se iz posredovanih podatkov odloči, ali je potrebna optimizacija. Izbere ustrezno rutino iz *HLIOL* ter aktivira dinamični prevajalnik in povezovalnik, ki v času izvajanja izvedeta vstavljeno, izbrano rutino.

Dinamična analiza prevajalnika za izvedbo rutine kolektivnega vhoda-izhoda (*collective I/O*) najprej določi vzorce dostopov do podatkov [3]. Nato določi vzorec shrambe teh podatkov. S primerjanjem vzorcev shrambe in dostopov se prevajalnik odloči, ali bo optimizacija s kolektivnim vhodom-izhodom izvedena. Nato dinamično spremeni kodo [3]. Pri rutini kolektivnega vhoda-izhoda želimo zmanjšati število dostopov do diskov. Bistvo je, da se več manjših zahtev združi in zmanjša celotno število zahtev [3].

V rutini kolektivnega vhoda-izhoda za primer branja najprej procesorji komunicirajo med seboj, da je znana količina podatkov [3]. Nato se določijo procesorji, ki bodo izvedli vhodno-izhodno zahtevo, da se minimizira količina dostopov. Podskupina procesorjev v imenu vseh (vzporedno) izvede zahtevo. Nazadnje se podatki z med-procesorsko komunikacijo prenesejo do procesorjev, ki jih zahtevajo [3].

Rutina pod-datotek (*sub-filing*) se sproži, ko se pogosto zahteva velika množica podatkov, toda uporabi se le manjši del iz te množice [3]. Da se prepreči prenos velikih datotek, trenutno uporabniki ročno razbijejo datoteke v manjše. Ogromna množica podatkov je pri *sub-filing* avtomatično razbita v manjše dele (*chunks*), ki se v shrambi shranijo kot pod-datoteke. To omogoči, da je potrebno prenesti le manjši kos od celote [3].

### 3.7.3 Profiliranje

Drugačen pristop k upravljanju razporeditve podatkov (*data layout*) med diski omogoča profiliranje aplikacije [3]. Diski poznajo stanje aktivnega dostopa (*active*), stanje neaktivnosti (*idle*) in po dalj časa neaktivnosti so upočasnjeni (*spin-down*) v stanje pripravljenosti (*standby*), ki je stanje nizke porabe električne energije [3]. Po potrebi se disk pospeši (*spin-up*) in vrne v stanje aktivnosti. To je najbolj preprost model upravljanja z energijo diskov na strežnikih [3]. Tako upočasnitev kot pospešitev (*spin-up*) zahtevata čas in energijo, zato želimo zmanjšati število le teh. Želimo tudi zakasniti čas uporabe diskov, da podaljšamo čas nizke porabe [3].

#### Razporejanje podatkov po diskih

Razporejanje datotek med diski (*file striping*) razbije velike datoteke na manjše delce in jih shranjuje na podskupino diskov v krožnem ponavljanju (*round-robin*). Trojček informacij (*start_disk*, *stipe_factor*, *stripe_size*) nam pove razpored podatkov med diski [3]:

- *Start_disk* pove identifikacijsko številko diska, na katerem se drobci datoteke začnejo.
- *Stipe_factor* pove število diskov, med katerimi se datoteka porazdeljuje.
- *Stripe_size* sporoči, kako velik je prostor za delec (*strip*) datoteke, ki se shrani v trenutku vpisa.

Cilj tega postopka je omogočiti vzporedne dostope do diskov in porazdelitev dostopov [3]. Več obstoječih datotečnih sistemov in vhodno-izhodnih knjižic za visoko-zmogljive sisteme ima *application programming interface* (*API*), ki posredujejo razpored podatkov ob kreaciji datoteke [3]. Za primer *Parallel virtual file system* (*PVFS*) omogoča uporabniku spremembo privzetih parametrov razporeda.

#### Izvedba

Najprej se v kodo aplikacije vstavijo ukazi za zbiranje informacij za vsak dostop do diskov. Rezultat je znano zaporedje dostopov z informacijami [3]:

- '$X_i$' za identiteto polja, ki je shranjen na disku,
- '$a_i$' za odmik do naslovljenega elementa polja in
- '$t_i$', ki je časovni odmik (*timestamp*) od začetka aplikacije, ki se zmanjša za čas vhodno-izhodne operacije (*access_time* - *disk_IO_time*).

Ko poznamo zaporedje (($X_1$, $a_1$, $t_1$), ($X_2$, $a_2$, $t_2$) ... ($X_n$, $a_n$, $t_n$)), izberemo razpored podatkov (*data layout*) med diski s porazdeljevanjem podatkov (*data striping*) med njimi tako, da minimiziramo konfliktne dostope znotraj in med polji [3]. Konflikti nastanejo pri dostopu znotraj polja (*intraarray*), ko dostopamo na isti disk (do istega polja), vendar do drugih elementov, v manjšem časovnem razmiku dostopov, kot je odzivni čas diska [3]. Primer: ($X_2$, $a_1$, 200 ms) in ($X_2$, $a_{10}$, 201 ms), odzivni čas diska R = 10 ms. Konflikti med polji (*interarray*) nastanejo pri dostopu različnih polj do istega diska v manjšem časovnem razmiku dostopov kot je odzivni čas diska [3]. Primer: ($X_2$, $a_1$, 100 ms) in ($X_4$, $a_1$, 108 ms), odzivni čas diska R = 10 ms.

Ob velikem številu diskov je iskalni prostor za izbiro nastavitev porazdelitve podatkov prevelik [3]. Profiliranje najprej izbere $stripe\_factor$, sledi *stripe_size* pod restrikcijo prejšnje izbire in nazadnje *start_disk*. V tem je profiliranje hitro hevristično iskanje nastavitev z nastavitvami blizu optimalnih v večini primerov [3].

### Primer za ilustracijo

Polja so shranjena v eni datoteki (niso razdrobljena) [3]. Imamo šest diskov, odzivni čas (*response time*) pet milisekund. Vsaka iteracija katerekoli izmed treh zank vzame deset milisekund [3].

Prva zanka:

```
for i=0 to 2047 {
    ... X[i] ...;
    ... X[i+1024] ...;
    ... Y[i] ...;
}
```

Druga zanka:

```
for i=0 to 1023{
    ... Z[i] ...;
    ... Z[i+256] ...;
    ... Z[i+512] ...;
}
```

Tretja zanka:

```
for i=0 to 4095{
    ... X[i] ...;
}
```

Po analizi dostopov v treh zankah profiliranje zaključi, da polje X potrebuje (najmanj) 2 diska, polje Y (najmanj) 1 disk in polje Z (najmanj) 3 diske [3]. Kot se opazi, gre za število dostopov iz teles zank, da bi omogočili vzporeden dostop na diske.

Recimo, da datotečni sistem podpira 256B, 512B, 1024B in 2048B velikosti delcev (*stripe size*) [3]. V prvi zanki, ker polje X porazdeljujemo med 2 diska, velikost delca 256B povzroči 2048 konfliktov, 512B povzroči 2048 konfliktov, 1024B nobenega in 2048B povzroči 1024 konfliktov, vsi so notranji. Izbere se velikost delca, ki povzroča najmanj konfliktov, to je 1024B za polje X [3]. Pri polju Z imamo: ob 256B nobenega konflikta, pri 512B jih je 1024, pri 1024B jih je 2048 in ob 2048B jih je 3072. Torej 256B delci so izbrani za polje Z [3]. Polje Y seveda izpustimo, ker je njegova datoteka v celoti na enem disku [3].

Začetni disk se izbere za posamezno polje na način, ki minimizira konflikte v dostopih različnih polj [3]. Polje X trivialno postavimo na disk 0. Polje Y, zaradi prve zanke izbira med preostalimi (tretji, četrti, peti ali šesti disk). Izberemo disk 2 (tretjega). Polje Z prav tako izbira diske, da ni konfliktov pri dostopih s polji X in Y [3]. Rezultat je lahko več razporedov.

| polje | *start_disk* | *stripe_factor* | *stripe_size* |
|---|---|---|---|
| X | 0 | 2 | 1024B |
| Y | 2 | 1 | 2048B |
| Z | 0 | 3 | 256B |

Vendar bi lahko ustvarili tudi ekvivalenten razpored z enakim stevilom konfliktov [3]:

| polje | *start_disk* | *stripe_factor* | *stripe_size* |
|---|---|---|---|
| X | 0 | 4 | 1024B |
| Y | 4 | 2 | 1024B |
| Z | 0 | 6 | 256B |

Vendar je alternativa slabša na področju energetske učinkovitosti. Pri prvotnem razporedu so aktivni največ trije diski in so v poskusu v viru [3] zahtevali šest prehodov v aktivno stanje. Alternativa je zahtevala trinajst. Kot sem omenil na začetku razdelka, preklop med stanji diska zahteva čas in energijo. Za grafično predstavitev vas vabimo, da si ogledate vir [3].

### 3.7.4 Varčevanje na komunikacijah

Arhitekture "omrežja na čipu" (*network-on-chip*) so se pojavile kot alternative vodilom od točke do točke. V viru [3] navajajo raziskave, kjer so ugotovili prihranke energije po kvadratni funkciji in linearni upad zmogljivosti komunikacijskih kanalov ob uravnavanju električnih napetosti oziroma frekvenc na njih.

Strojni nadzor porabe električne energije v omrežjih na čipu preverja obremenitev posameznega kanala in nato ustrezno nastavi električno napetost oziroma frekvenco za komunikacijski kanal [3]. Ob neuporabi ga izklopi. Ta pristop je najbolj študiran, vendar naj bi s prevajalniki potencialno dosegli boljše rezultate s statično analizo komunikacij in statično (vnaprej) nastavili napetosti komunikacijskih kanalov [3].

#### Z uporabo zbrane statistike

V viru [3] citirajo pristop z uporabo zgodovine uporabe komunikacijskih kanalov. Namenjen je dvodimenzionalni mrežni arhitekturi s statičnim algoritmom za iskanje poti (*routing*) skoznjo. Omrežje je dostopno prevajalniku skozi vmesnik (*interface*) [3]. Cilj je, da prevajalnik spremeni kodo programa in upravlja z omrežjem skozi vmesnik z uporabo uravnavanja napetosti (oziroma frekvence) [3].

Ko se izvaja gnezdo zank aplikacije, se zbere statistika uporabe komunikacijskih kanalov v zgodnjih iteracijah (med izvajanjem aplikacije) [3]. Iz te se izračuna dovoljena zakasnitev komunikacije in pasovne širine za komunikacijske kanale, ki ne bodo delovali pri polni zmogljivosti. Na podlagi izračunov se nato izbere ustre-

zna električna napetost oziroma frekvenca za komunikacijske kanale [3]. Preostale iteracije gnezda zank se izvajajo pod nastavljenimi omejitvami komunikacij. Algoritem temelji na pojavu komunikacijskih vzorcev v sledečih iteracijah zank [3].

### S statično analizo

Drugi pristop je namenjen vzporednim aplikacijam, ki uporabljajo izmenjavo sporočil (*message passing comunication*) [3]. Med-procesorska komunikacija naj bi bila že optimizirana, na primer z vektorizacijo (*message vectorization*) in združevanjem (*message coalescing*) sporočil [3]. In procesi morajo biti že dodeljeni k procesorjem.

Koda, ki ustreza zgoraj naštetim pogojem, se analizira in izdela se graf komunikacije med procesi (lahko berete tudi: "med procesorji") [3]. Nato se z analizo kritične poti v grafu določijo ustrezni koraki uravnavanja (velikost preskoka ob znižanju/povečanju električne napetosti) in nastavi se začetna električna napetost za komunikacijske kanale [3]. Nazadnje prevajalnik le še vstavi eksplicitne klice za upravljanje s porabo v kodo.

Ta predlog prevajalnika iz vira [3] izkorišča razlike v količini podatkov, poslanih v sporočilu ali v količini dela med pošiljanjem in sprejemom za varčevanje s porabo električne energije na komunikacijskih kanalih. Program se ne ustavlja pri ukazu za pošiljanje (ne-blokirajoča oblika). Ustavi se pri ukazu za sprejem (blokirajoč) [3]. Primera:

- Prvi procesor pošlje veliko sporočilo (1024 objektov) drugemu, nato čaka na sporočilo od drugega. Drugi pošlje manj podatkov (256 objektov) prvemu in čaka na sporočilo prvega. V tem primeru znižamo napetost na komunikacijskem kanalu od drugega procesorja k prvemu, saj vsebuje manj podatkov in se lahko pošiljajo počasneje [3].

- Prvi procesor pošlje sporočilo drugemu. Nato kratek čas procesira in čaka na sporočilo drugega. Drugi procesor pošlje enako veliko sporočilo prvemu. Nato dolgo časa procesira in čaka na sporočilo prvega. V tej situaciji znižamo napetost na komunikacijskem kanalu od prvega k drugemu, saj bo drugi procesor dalj časa zaposlen s procesiranjem, preden bo pripravljen sprejeti sporočilo [3].

# Poglavje 4

# Zaključek

Kot je postalo jasno že na začetku v razdelku, namenjenem ogljičnemu odtisu, predstavlja celoten sektor informacijskih tehnologij le 2 % do 2,5 % celoletnega ogljičnega odtisa sveta. Vpliv informacijskih tehnologij na podnebne spremembe je torej majhen.

Če vas zanima energetska učinkovitost osebnih računalnikov, moramo priznati, da so ti že energetsko učinkoviti z omejitvami. S tem merim na uhajanje elektronov skozi ubežni tok (*leakage current*) v tranzistorjih. Vendar bi v tem smislu diplomo lahko preprosto zaključil s stavkom: "Počakajte na izum boljših tranzistorjev". Sedaj lahko trdim, da imamo že sedaj mehanizme, s katerimi lahko kontroliramo porabo električne energije .

Namizni računalniki in prenosne napravice že uporabljajo sisteme za varčevanje z energijo. Novost je vpeljava varčevanja z energijo na velike računalniške sisteme, na superračunalnike, kjer se doslej z varčevanjem niso trudili. Mehanizme varčevanja s prenosnih naprav z novimi načini uporabe teh mehanizmov lahko vpeljemo za superračunalnike. Prilagoditve so trenutno predmet raziskav. S tem smo v diplomi dosegli pregled področja in uspeli prikazati nekaj trenutnih predlogov za računalniške sisteme velikih razsežnosti.

Podobne manjše, evolucijske izboljšave so tudi na drugih napravah na področju energetske učinkovitosti. Izjema so organske, ploščate LED svetilke in druge naprave, kjer vključitev organskih materialov predstavlja večjo novost.

Vsak razdelek zelenega računalništva, razen razdelka o dinamičnem uravnavanju frekvence procesorja (3.2), je prevod izsledkov neodvisnih raziskav s področja

zelenega računalništva, kjer so bili rezultati že normalizirani in predstavljeni neodvisno od sistema. Zato ne morem navesti konkretnih porab (koliko vatov) in prihrankov ob vpeljavi algoritma ali sistema iz posameznega poglavja in še manj kombinacije teh. Zadovoljiti smo se morali zgolj z manjšim uspehom v spoznanju teh mehanizmov.

V splošnem, pri problemu podnebnih sprememb, strokovnjaki za podnebne spremembe, okoljski aktivisti in tudi strokovnjaki drugih naravoslovnih znanosti svarijo, da se čas za omejitev posledic podnebnih sprememb izteka. Svetovne vlade naj se ne opirajo na slepo upanje, da bo izumljena "magična paličica", ki bo problem enostavno odpravila. S tem dokumentom upam, da smo bralca prepričali, da vzroki za slabo reševanje problema podnebnih sprememb nikakor niso v pomanjkanju tehnologij ali idej. S tem sem prisiljen zaključiti, da podnebne spremembe niso tehnološki problem.

# Slike

# Literatura


[1] *The Green Computing Book: Tackling Energy Efficiency at Large Scale*, Chapman & Hall/CRC Computational Science, ur. Wu-Chung Feng. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2014.

[2] M. Curtis-Maury, D. S. Nikolopoulus, "*Energy-Efficient Multithreading through Run-Time Adaptation*," v *The Green Computing Book: Tackling Energy Efficiency at Large Scale*, Chapman & Hall/CRC Computational Science, ur. Wu-Chung Feng. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2014, str. 115-148.

[3] M. Kandemir, S. Srikantaiah, "*Compiler-Driven Energy Efficiency*," v *The Green Computing Book: Tackling Energy Efficiency at Large Scale*, Chapman & Hall/CRC Computational Science, ur. Wu-Chung Feng. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2014, str. 43-86.

[4] Z. Wang, P. Ranganathan, "*Cross-Layer Power Management*," v *The Green Computing Book: Tackling Energy Efficiency at Large Scale*, Chapman & Hall/CRC Computational Science, ur. Wu-Chung Feng. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2014, str. 183-229.

[5] C. Hsu, W. Feng, S. W. Poole, "*An Adaptive Runtime System for Improving Energy Efficiency*," v *The Green Computing Book: Tackling Energy Efficiency at Large Scale*, Chapman & Hall/CRC Computational Science, ur. Wu-Chung Feng. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2014, str. 87-113.

[6] A. R. Butt et al., "*Exploring Trade-Offs between Energy Savings and Reliability in Storage Systems*," v *The Green Computing Book: Tackling Energy*

*Efficiency at Large Scale*, Chapman & Hall/CRC Computational Science, ur. Wu-Chung Feng. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2014, str. 149-182.

[7] IBM Blue Gene Team, "*Low-Power, Massively Parallel, Energy-Efficient Supercomputers*," v *The Green Computing Book: Tackling Energy Efficiency at Large Scale*, Chapman & Hall/CRC Computational Science, ur. Wu-Chung Feng. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2014, str. 1-41.

[8] B. Beaumont-Thomas. (2015). Smash-hit Chinese pollution doc Under the Dome taken offline by government. Dostopno na: http://www.theguardian.com/film/2015/mar/09/chinese-pollution-documentary-under-the-dome-taken-offline-government

[9] M. Dale, S. M. Benson. (2013). Energy balance of the global photovoltaic (PV) industry-is the PV industry a net electricity producer?, *Environmental science & technology*. 47(7), str. 3482-3489. Dosegljivo na: ACS Publications, http://pubs.acs.org/doi/abs/10.1021/es3038824

[10] H. S. Dunn. (2010). The carbon footprint of ICTs. Dostopno na: http://www.giswatch.org/thematic-report/sustainability-climate-change/carbon-footprint-icts

[11] B. Garside. (2014). Global carbon emissions rise to new record in 2013: report. Dostopno na: http://www.reuters.com/article/2013/11/19/us-global-carbon-emissions-idUSBRE9AI00A20131119

[12] J. Gwiazda, F. A. DiBella. (2009). Pit Power Tower. Dostopno na: http://www.altenergymag.com/emagazine.php?issue_number=09.02.01&article=gwiazda

[13] J. Kaiman. (2015). China's reliance on coal reduces life expectancy by 5.5 years, says study. Dostopno na: http://www.theguardian.com/environment/2013/jul/08/northern-china-air-pollution-life-expectancy

[14] S. Kirchgaessner. (2015). Pope's climate change encyclical tells rich nations: 'pay your debt to the poor'. Dostopno na: http:

//www.theguardian.com/world/2015/jun/18/popes-climate-change-encyclical-calls-on-rich-nations-to-pay-social-debt

[15] R. Natarajan. (2011). Raid 0, raid 1, raid 5, raid 10 explained with diagrams. Dostopno na: http://www.thegeekstuff.com/2010/08/raid-levels-tutorial/

[16] R. Natarajan. (2011). Raid 2, raid 3, raid 4, raid 6 explained with diagram. Dostopno na: http://www.thegeekstuff.com/2011/11/raid2-raid3-raid4-raid6/

[17] G. Readfearn. (2015). How fossil fuel emissions could take protein from the diets of the world's poorest people. Dostopno na: http://www.theguardian.com/environment/planet-oz/2015/jun/17/how-fossil-fuel-emissions-could-take-protein-from-the-diets-of-the-worlds-poorest-people

[18] M. Safi. (2015). Tesla announces low-cost batteries for homes. Dostopno na: http://www.theguardian.com/technology/2015/may/01/tesla-announces-low-cost-solar-batteries-elon-musk

[19] PBL Netherlands Environmental Assesment Agency. (2013). Trends in global CO2 emissions: 2013 Report. *The Emissions Database for Global Atmospheric Research (EDGAR)*. Dostopno na: http://edgar.jrc.ec.europa.eu/news_docs/pbl-2013-trends-in-global-co2-emissions-2013-report-1148.pdf

[20] (2015) 2009 United Nations Climate Change Conference - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=2009_United_Nations_Climate_Change_Conference&oldid=662283704

[21] (2015) 3d processors, memory and storage explained. Dostopno na: http://www.techradar.com/news/computing-components/3d-processors-memory-and-storage-explained-987509/1

[22] (2015) Advanced configuration and power interface — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Advanced_Configuration_and_Power_Interface&oldid=671170407

[23] (2015) AMD FX-Series FX-8350 - FD8350FRW8KHK / FD8350FRHKBOX. Dostopno na: http://www.cpu-world.com/CPUs/Bulldozer/AMD-FX-Series%20FX-8350.html

[24] (2015) Bibavica — Wikipedija, prosta enciklopedija. Dostopno na: http://sl.wikipedia.org/w/index.php?title=Bibavica&oldid=4468200

[25] (2015) Biomass — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Biomass&oldid=668500120

[26] (2016) Minas Basin Pulp and Power Marine Current Turbines have partnered on a tidal test project in Nova Scotia — Flickr - Photo Sharing. Dostopno na: http://www.flickr.com/photos/greenenergyfutures/8248501200

[27] (2015) Carbon footprint - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Carbon_footprint&oldid=648776826

[28] (2015) Charts, benchmarks cpu charts 2015, [35] system peak power. Dostopno na: http://www.tomshardware.co.uk/charts/cpu-charts-2015/-35-System-Peak-Power,Marque_fbrandx32,3727.html

[29] (2016) Commons:GNU Free Documentation License, version 1.2 - Wikimedia Commons. Dostopno na: http://commons.wikimedia.org/wiki/Commons:GNU_Free_Documentation_License,_version_1.2

[30] (2015) Compact linear Fresnel reflector — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Compact_linear_Fresnel_reflector&oldid=604113790

[31] (2015) Concentrated solar power — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Concentrated_solar_power&oldid=615195164

[32] (2015) Concentrated Solar Power. Dostopno na: http://solarcellcentral.com/csp_page.html

[33] (2016) Creative Commons Legal Code 2.0 NonCommercial. Dostopno na: http://creativecommons.org/licenses/by-nc/2.0/legalcode

[34] (2016) Creative Commons Legal Code 2.0 NonComercial, ShareAlike. Dostopno na: https://creativecommons.org/licenses/by-nc-sa/2.0/legalcode

[35] (2016) Creative Commons Legal Code 2.0 ShareAlike. Dostopno na: http://creativecommons.org/licenses/by-sa/2.0/legalcode

[36] (2016) Creative Commons Legal Code 3.0. Dostopno na: http://creativecommons.org/licenses/by-sa/3.0/legalcode

[37] (2015) Crystalline silicon — Wikipedia, The Free Encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Crystalline_silicon&oldid=690727682.

[38] (2015) Dynamic compilation — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Dynamic_compilation&oldid=684534140

[39] (2015) Dynamic frequency scaling — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Dynamic_frequency_scaling&oldid=661010293

[40] (2016) File:Novatec Solar Puerto Errado 2 BoP PI.jpg - Wikimedia Commons. Dostopno na: http://commons.wikimedia.org/w/index.php?title=File:Novatec_Solar_Puerto_Errado_2_BoP_PI.jpg&oldid=93316448

[41] (2016) File:Parabolic trough solar thermal electric power plant 1.jpg - Wikimedia Commons. Dostopno na: http://commons.wikimedia.org/w/index.php?title=File:Parabolic_trough_solar_thermal_electric_power_plant_1.jpg&oldid=183498092

[42] (2016) Gemasolar solar thermal power plant, Spain — Flickr - Photo Sharing. Dostopno na: http://www.flickr.com/photos/beyondcoalandgas/9296666273

[43] (2015) Geothermal electricity — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Geothermal_electricity&oldid=669492861

[44] (2016) Grid-connected photovoltaic power system — Wikipedia, The Free Encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Grid-connected_photovoltaic_power_system&oldid=699899556

[45] (2015) High-altitude wind power — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=High-altitude_wind_power&oldid=665281075

[46] (2015) Hydroelectric power generation. Dostopno na: http://www.mpoweruk.com/hydro_power.htm

[47] (2015) Hydroelectricity — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Hydroelectricity&oldid=668090283

[48] (2015) Ionic liquid — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Ionic_liquid&oldid=611916116

[49] (2015) IPCC - Intergovernmental Panel on Climate Change. Dostopno na: http://ipcc.ch/organization/organization_history.shtml

[50] (2015) Kyoto Protocol - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Kyoto_Protocol&oldid=667846384

[51] (2015) Parabolic trough — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Parabolic_trough&oldid=614184143

[52] (2015) Parabolic through technology. Dostopno na: http://www.schott.com/csp/english/parabolic-through-technology.html

[53] (2015) Processor boosting control. Dostopno na: http://www.kernel.org/doc/Documentation/cpu-freq/boost.txt

[54] (2015) Sahara's solar power potential underlined. Dostopno na: http://social.csptoday.com/taxonomy/term/83/saharas-solar-power-potential-underlined

[55] (2015) Sata unrecoverable errors and how that impacts raid. Dostopno na: http://subnetmask255x4.wordpress.com/2008/10/28/sata-unrecoverable-errors-and-how-that-impacts-raid/

[56] (2016) Solar Concentrator Array — Flickr - Photo Sharing. Dostopno na: http://www.flickr.com/photos/brewbooks/6263643186

[57] (2014) Solar dish/engine system. Dostopno na: http://theenergylibrary.com/node/451

[58] (2015) Solar energy — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Solar_energy&oldid=615625752

[59] (2015) Solar panel — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Solar_panel&oldid=614966532

[60] (2015) Solar Panels – Just how do they Work? Dostopno na: http://dfwsolarelectric.com/solar-panels-how/

[61] (2016) Solar power — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Solar_power&oldid=701490177

[62] (2015) Solar power tower — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Solar_power_tower&oldid=611979242

[63] (2015) Solar thermal energy — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Solar_thermal_energy&oldid=614857100

[64] (2015) Solar updraft tower — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Solar_updraft_tower&oldid=612378409

[65] (2015) Sources of Greenhouse Gas Emissions. Dostopno na: http://www3.epa.gov/climatechange/ghgemissions/global.html#four

[66] (2015) Standard raid levels — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Standard_RAID_levels&oldid=654032781

[67] (2016) Stirling Energy Systems - Dish Stirling Engine [found] — Flickr - Photo Sharing. Dostopno na: http://www.flickr.com/photos/derek_curry/49083646

[68] (2016) Stirling engine — Wikipedia, The Free Encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Stirling_engine&oldid=700764772

[69] (2015) The Most Frequently Asked Questions about Wind Energy. Dostopno na: http://www.culturechange.org/wind.htm

[70] (2015) Tidal power — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Tidal_power&oldid=667568341

[71] (2015) Wave power — wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Wave_power&oldid=666306763

[72] (2015) Wind power — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/w/index.php?title=Wind_power&oldid=618042592

[73] (2015) World Record Solar Cell with 44.7% Efficiency. Dostopno na: http://www.ise.fraunhofer.de/en/press-and-media/press-releases/presseinformationen-2013/world-record-solar-cell-with-44.7-efficiency